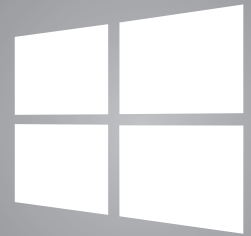


msdn magazine



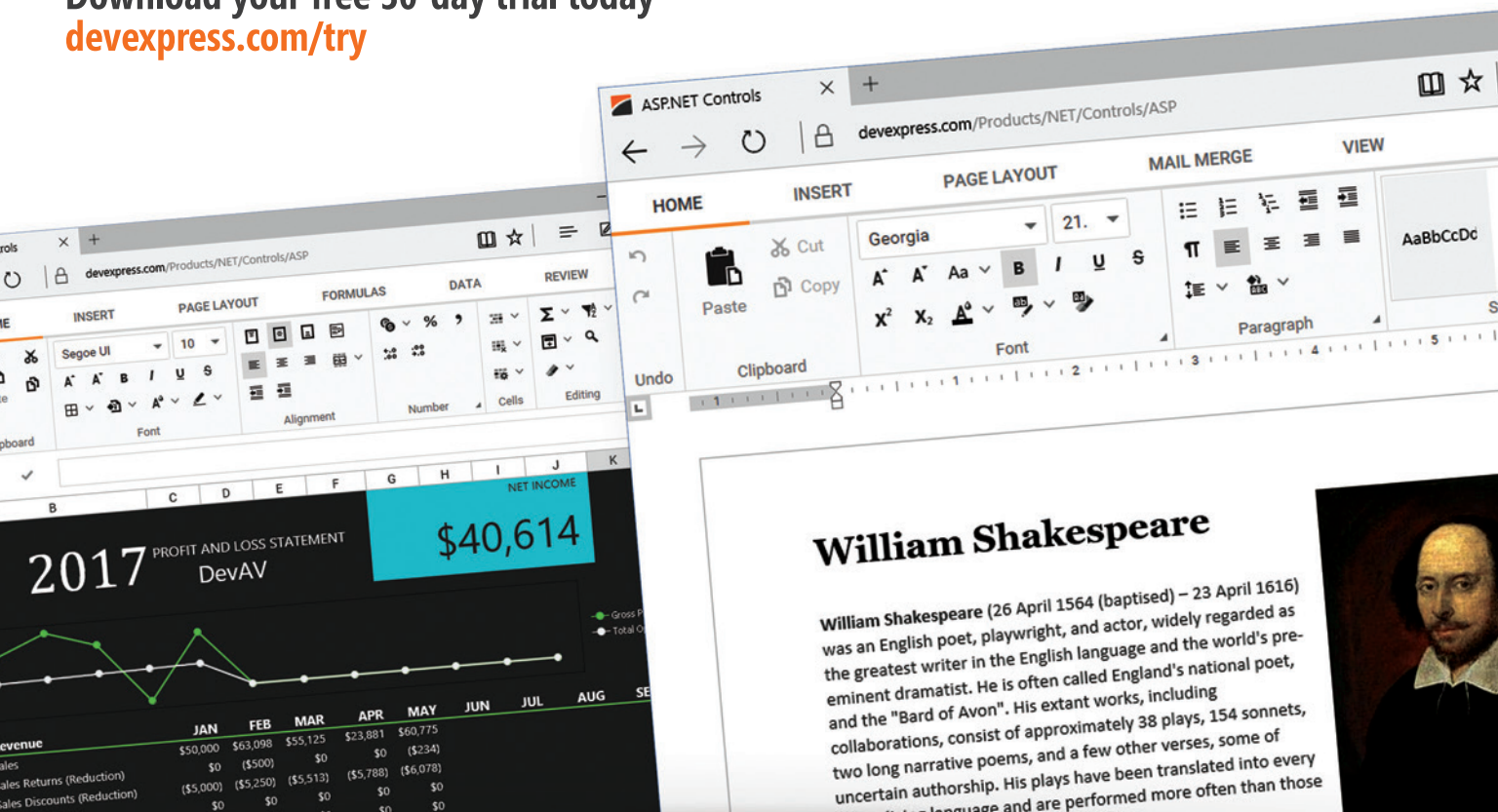
Special Issue

Office-Inspired ASP.NET & MVC Controls

Create high-impact line-of-business applications for the web with the DevExpress ASP.NET Subscription.



Download your free 30-day trial today
devexpress.com/try





Your Next Great Web App Starts Here

From apps that replicate the look and feel of Microsoft Office® 365, to high-impact decision support systems for your enterprise, DevExpress Web Controls for ASP.NET will help you build your best, without limits or compromise.



Download your free 30-day trial
and experience the DevExpress difference today.

devexpress.com/try

msdn magazine



Special Issue

Connect(); Special Issue

Getting Started with Microsoft AI Joseph Sirosh and Wee Hyong Tok	6
Deliver On-Device Machine Learning Solutions Larry O'Brien	14
Introducing App Center Matt Gibbs	24
The Road to Continuous Delivery with Visual Studio Team Services Willy-Peter Schaub	34
Create Serverless APIs Using Azure Functions Alex Karcher	42
SQL Operations Studio: Cross-Platform SQL Server Management Julie Lerman	50
Introducing the Windows Compatibility Pack for .NET Core Immo Landwerth	60



Infragistics Ultimate 17.2

Productivity Tools & Fast Performing UI Controls for Quickly Building Web, Desktop, & Mobile Apps

Includes 100+ beautifully styled, high-performance grids, charts & other UI controls, plus visual configuration tooling, rapid prototyping, and usability testing.

Angular | JavaScript / HTML5 | ASP.NET | Windows Forms | WPF | Xamarin

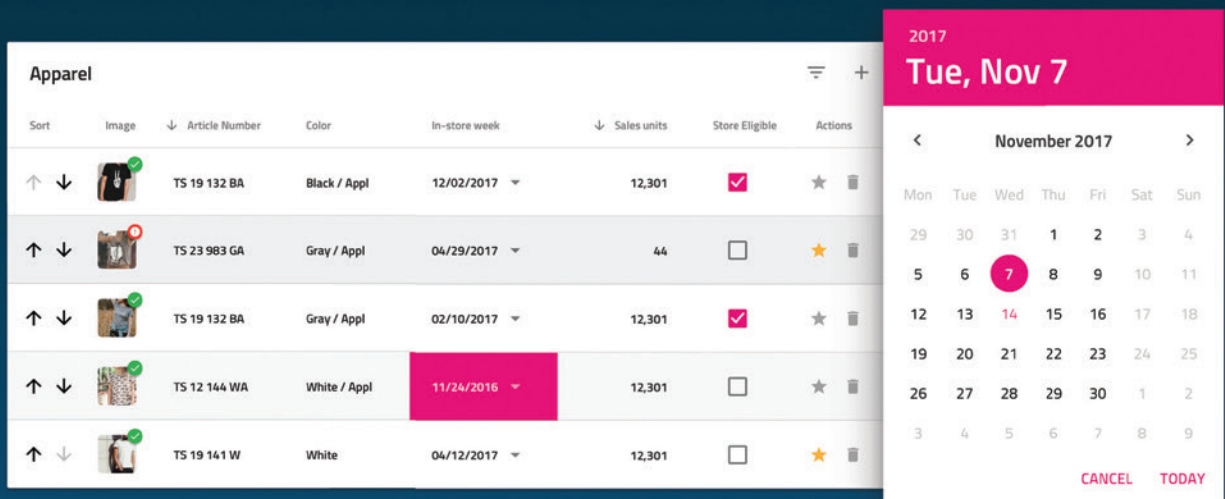
Download a free trial at
Infragistics.com/Ulimate



Featuring

Ignite UI

A complete UI component library for building high-performance, data rich web applications



- ✓ Create beautiful, touch-first, responsive desktop & mobile web apps with over 100 JavaScript / HTML5, MVC & **Angular components**.
- ✓ Our easy to use Angular components have no 3rd party dependencies, a tiny footprint, and easy-to-use API.
- ✓ The Ignite UI **Angular Data Grid** enables you to quickly bind data with little coding - including features like sorting, filtering, paging, movable columns, templates and more!
- ✓ Speed up development time with responsive layout, powerful data binding, cross-browser compatibility, WYSIWYG page design, & built-in-themes.

Download a free trial of Ignite UI at: **Infragistics.com/ignite-ui**

To speak with our sales team or request a product demo call: 1.800.321.8588

General Manager Jeff Sandquist

Director Dan Fernandez

Editorial Director Jennifer Mashkowski mmeditor@microsoft.com

Site Manager Kent Sharkey

Editorial Director, Enterprise Computing Group Scott Bekker

Editor in Chief Michael Desmond

Features Editor Sharon Terdeman

Group Managing Editor Wendy Hernandez

Senior Contributing Editor Dr. James McCaffrey

Contributing Editors Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt

Vice President, Art and Brand Design Scott Shultz

Art Director Joshua Gould



President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

ART STAFF

Creative Director Jeffrey Langkau
Associate Creative Director Scott Rovin
Senior Art Director Deirdre Hoffman
Art Director Michele Singh
Art Director Chris Main
Senior Graphic Designer Alan Tao
Senior Web Designer Martin Peace

PRODUCTION STAFF

Print Production Manager Peter B. Weller
Print Production Coordinator Lee Alexander

ADVERTISING AND SALES

Chief Revenue Officer Dan LaBianca
Regional Sales Manager Christopher Kourtoglou
Advertising Sales Associate Tanya Egenolf

ONLINE/DIGITAL MEDIA

Vice President, Digital Strategy Becky Nagel
Senior Site Producer, News Kurt Mackie
Senior Site Producer Gladys Rama
Site Producer Chris Paoli
Site Producer, News David Ramel
Director, Site Administration Shane Lee
Front-End Developer Anya Smolinski
Junior Front-End Developer Casey Rysavy
Office Manager & Site Assoc. James Bowling

LEAD SERVICES

Vice President, Lead Services Michele Imgrund
Senior Director, Audience Development & Data Procurement Annette Levee
Director, Audience Development & Lead Generation Marketing Irene Fincher
Director, Client Services & Webinar Production Tracy Cook
Director, Lead Generation Marketing Eric Yoshizuru
Director, Custom Assets & Client Services Mallory Bastionell
Senior Program Manager, Client Services & Webinar Production Chris Flack
Project Manager, Lead Generation Marketing Mahal Ramos

ENTERPRISE COMPUTING GROUP EVENTS

Vice President, Events Brent Sutton
Senior Director, Operations Sara Ross
Senior Director, Event Marketing Merikay Marzoni
Events Sponsorship Sales Danna Vedder
Senior Manager, Events Danielle Potts
Coordinator, Event Marketing Michelle Cheng
Coordinator, Event Marketing Chantelle Wallace



Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Chief Financial Officer
Craig Rucker

Chief Technology Officer
Erik A. Lindgren

Executive Vice President
Michael J. Valenti

Chairman of the Board
Jeffrey S. Klein

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o MSDN Magazine, 4 Venture, Suite 150, Irvine, CA 92618.

LEGAL DISCLAIMER The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International
Phone: 212-221-9595
E-mail: 1105reprints@parsintl.com
Web: 1105Reprints.com

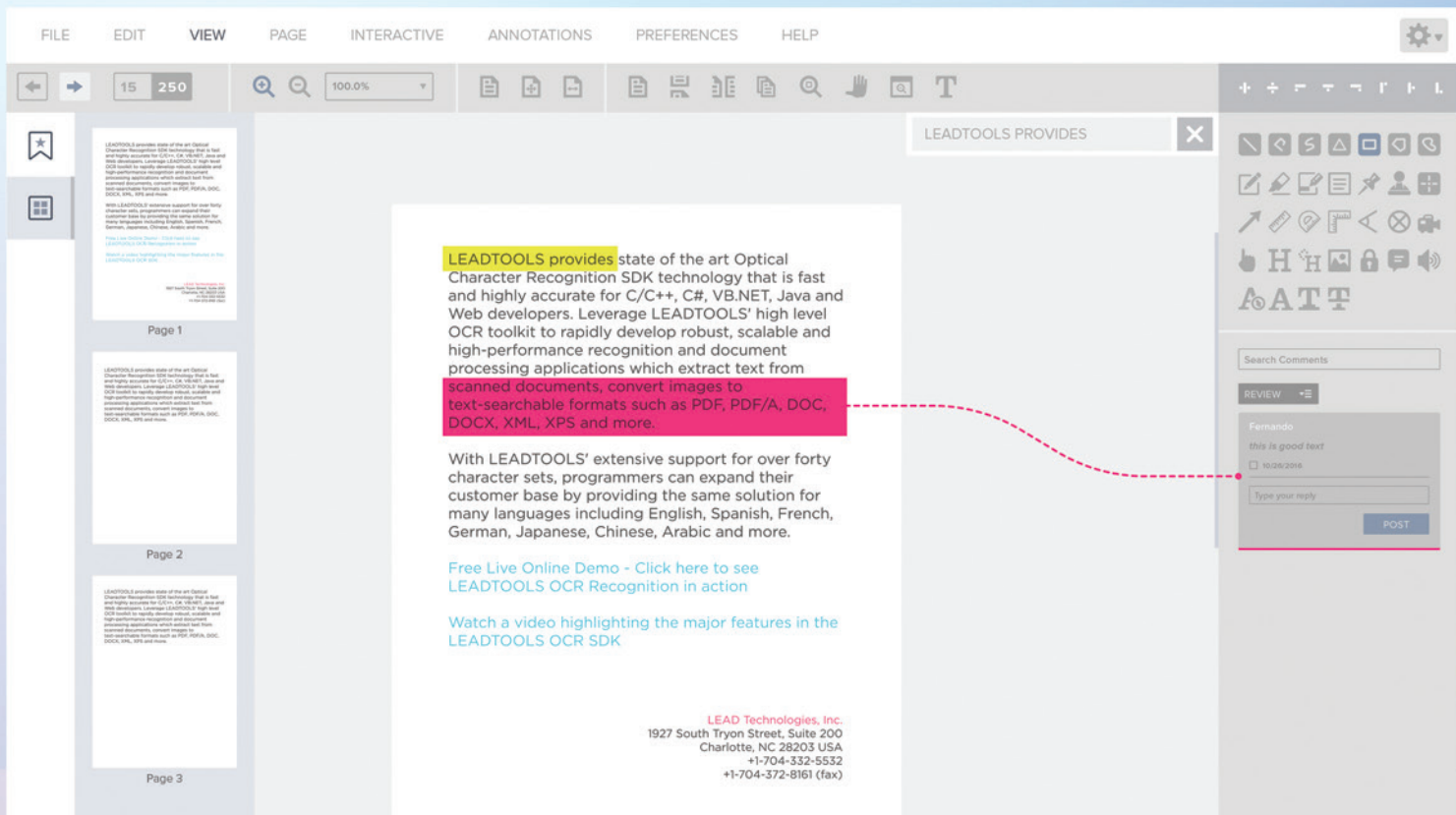
LIST RENTAL This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jljong@meritdirect.com; Web: meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. E-mail: To e-mail any member of the staff, please use the following form: FirstInitialLastName@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311
The opinions expressed within the articles and other contents herein do not necessarily express those of the publisher.



The leading Document Viewer SDK just got better.



With only a few lines of code, developers can use the **LEADTOOLS** HTML5/JavaScript Document Viewer SDK to add rich document viewing features to any project, including text search, annotation, memory-efficient paging, inertial scrolling, and vector display.



CREATE, ORGANIZE & EXPORT A DOCUMENT FROM MULTIPLE FILES



PDF, DOC, DOCX, RTF, HTML, SVG, and XPS



ANNOTATIONS & MARKUP



DRAG AND DROP INTERFACE



SEARCHABLE TEXT



SDKs for
WEB



SDKs for
MOBILE



SDKs for
DESKTOP



SDKs for
SERVER

.NET Windows API Java WinRT Linux iOS macOS Android JavaScript





Coming Up Connect();

Any developer, any application, any platform. It's been a rallying cry of the Microsoft Connect(); conference for years. From open sourcing key technologies like .NET Core, to unveiling important cross-platform tools like Visual Studio for Mac, to enabling platform-agnostic DevOps with Visual Studio App Center, the message has been as powerful as it has been consistent.

At Connect(); 2017 in November, Microsoft took its anyone, anyway, anywhere strategy and applied it to the fast-moving world of artificial intelligence (AI) and machine learning (ML). At the event, Microsoft Executive Vice President Scott Guthrie offered a preview of the Visual Studio Tools for AI extension, which enables developers and data scientists to create, train, manage, and deploy AI models that support deep learning frameworks like Cognitive Services, TensorFlow, and Caffe. He also previewed Azure IoT Edge, a service that deploys cloud intelligence to Internet of Things (IoT) devices via containers. Developers can build and test container-based workloads using C, Java, .NET, Node.js, and Python to leverage resources like Azure Machine Learning (Azure ML), Azure Functions, and Azure Stream Analytics.

That focus on AI has shaped our coverage in this special issue of *MSDN Magazine* focused on the Connect(); 2017 conference. The issue opens with a feature article written by Joseph Sirosh, corporate vice president of the Cloud AI Platform at Microsoft, and Wee Hyong Tok, principal data scientist manager of the Cloud AI Platform. Titled "Getting Started with Microsoft AI," the article offers a great overview of Microsoft's effort to help developers and data scientists leverage large-scale data flows and manage powerful computational models to spot trends, predict outcomes and speed decision making. Larry O'Brien follows with "Deliver On-Device Machine Learning Solutions," where he explores how AI and ML can be leveraged in the field on smartphones and devices.

Even as Connect(); 2017 explored the burgeoning AI space, it's remained grounded in the concepts that defined it from the beginning in 2014—developer productivity and open, cross-platform development. At these earlier events we've seen Microsoft announce open source initiatives and support; unveil tooling for

Mac, Linux, Android and iOS; and provide integrated solutions to streamline, automate and improve DevOps.

This year's conference showcased these same themes, and our coverage reflects that. The feature by Matt Gibbs, "Introducing App Center," looks at the newly released suite of cloud services that acts as "mission control" for DevOps. App Center enables beta distribution and crash analytics, app testing on physical devices, and cloud build services for continuous integration. Willy Peter Schaub then dives further into the process of DevOps with his article, "The Road to Continuous Delivery with Visual Studio Team Services."

At Connect(); 2017 in November,
Microsoft took its anyone,
anyway, anywhere strategy and
applied it to the fast-moving
world of artificial intelligence and
machine learning.

Developer productivity is highlighted in our coverage of C# and .NET. Immo Landwerth pens "Introducing the Windows Compatibility Pack for .NET Core," while Stephen Toub plumbs the new Span feature in C# 7.2 in "All About Span: Exploring a New .NET Mainstay," which appears as a Web feature for the special issue.

There's lots more, including explorations of Microsoft Graph, Azure Functions and the new SQL Operations Studio. In addition to the seven feature articles in the print issue of the magazine, check out the bonus articles that appear as part of the issue at msdn.com/magazine/mt814798.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2017 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.



DOCXCONVERTER
For Windows

Free Demo at DOCXConverter.com

Amyuni DOCX Converter for Windows

Convert any document, including PDF documents, into DOCX format.
Enable editing of documents using Microsoft Word or other Office products.



A standalone desktop version, a server product for automated processing or an SDK for integration into third party applications.

Create

Create naturally editable DOCX documents with paragraph formatting and reflow of text

Convert

Convert images and graphics of multiple formats into DOCX shapes

OCR

Use OCR technology to convert non-editable text into real text

Extract

Extract headers and footers from source document and save them as DOCX headers and footers

Open

Open PDF documents with the integrated PDF viewer and quickly resave them to DOCX format

Configure

Configure the way the fonts are embedded into the DOCX file for optimal formatting

A virtual printer driver available for Windows 7 to Windows 10 and Windows Server 2008 to 2016

Powered by Amyuni Technologies:

Developers of the Amyuni PDF Converter and Amyuni PDF Creator products integrated into hundreds of applications and installed on millions of desktops and servers worldwide.

www.docxconverter.com

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.

Getting Started with Microsoft AI

Joseph Sirosh and Wee Hyong Tok

Software developers are quickly adopting Artificial Intelligence (AI) technologies, such as natural language understanding, sentiment analysis, speech recognition, image understanding and machine learning (ML). Across a broad range of industries and sectors, AI-infused software applications and cloud services drive innovative customer experiences, augment human capabilities and transform how we live, work and play. New tools, cloud-hosted APIs and platforms make it even easier to build such applications.

Modern AI applications live at the intersection of cloud computing, data platforms and AI tools. The cloud provides a powerful foundation for elastic compute and storage, while supporting special-purpose hardware such as graphics processing units (GPUs) that accelerate demanding calculations. It also enables connectivity, identity, application monitoring and the Internet of Things

(IoT). Data platforms in the cloud can ingest and integrate massive volumes of data, use databases and data lakes to transform and analyze the data, and build real-time data-driven applications. Layered upon these capabilities are AI tools and algorithms that help developers build models from the data for targeted intelligent scenarios, and deploy them in a hosted AI application.

What are some examples of such applications? Here are a few examples:

- **Health Care:** Doctors in the Microsoft Intelligent Network for Eyecare (MINE) leverage AI to improve patient outcomes for eye surgeries by identifying optimal surgical parameters to personalize treatment and maximize the probability of success.

At the Cochrane Transform Project, AI is used to improve and streamline the nonprofit organization's comprehensive reviews of health care interventions, with the goal of identifying the best treatments and interventions for patients. AI is used to analyze thousands of research studies, sharply reducing the time spent on manual reviews and freeing staff to focus on more urgent tasks.

- **Manufacturing:** Jabil, one of the world's leading design and manufacturing companies, uses AI to optically inspect printed circuit boards and detect manufacturing defects. AI is also used in the manufacturing environment to monitor equipment to predict declining efficiency and impending failure of machines.

This article discusses:

- Getting started with Microsoft AI
- Overview of what's new in Azure Machine Learning
- Building an intelligent application using Cognitive Services
- Building an intelligent bot using Bot Framework

Technologies discussed:

Microsoft AI, Azure Machine Learning, Bot Framework, Cognitive Services

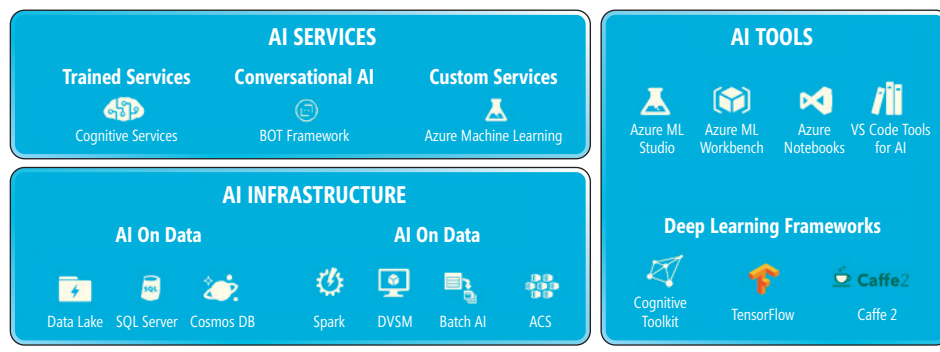


Figure 1 The Microsoft AI Platform

- **Retail:** Lowe's partnered with Microsoft to create a kitchen remodeling design experience driven by AI. The Lowe's customer shares his or her dream kitchen photos with a design specialist, who uses an AI-powered application to gain deep insight into the style and preferences of the customer. These findings are used to generate a match from the Lowe's dream kitchen collection, which is then shown in a mixed-reality environment so the customer can interact with the products (such as cabinets, counter tops and appliances). Like Lowe's, many retailers are turning to AI to re-imagine the retail experience for consumers.

interoperable services, APIs, libraries, frameworks and tools that developers can leverage to build smart applications. The Microsoft AI platform consists of three core areas: AI Services, AI Infrastructure and AI Tools. Let's start with a quick tour of the AI platform, shown in **Figure 1**.

Cognitive Services: Trained services like Cognitive Services enable you to jumpstart development of your AI applications, without requiring you or your data science team to develop and train the models. Cognitive Services features a rich set of instant AI capabilities that you can use. These AI capabilities are organized into the following categories: vision, speech, language, knowledge and search.

To help eliminate the heavy lifting involved in building end-to-end systems, Microsoft provides a powerful AI platform, composed of a set of loosely coupled but highly interoperable services on Azure.

Building these kinds of AI applications requires the integration of numerous components. To help eliminate the heavy lifting involved in building end-to-end systems, Microsoft provides a powerful AI platform, composed of a set of loosely coupled but highly interoperable services on Microsoft Azure. This article provides an overview of this platform and points to valuable resources for getting started developing exciting AI applications.

The Cognitive Services APIs enable you to leverage powerful computer vision algorithms that have been pre-trained to recognize things like different face attributes, landmarks, celebrities, gender, emotion, and printed or written words (Optical Character Recognition, or OCR). Powerful language capabilities can recognize commands from users, analyze key phrases, perform translations and spell check, and more.

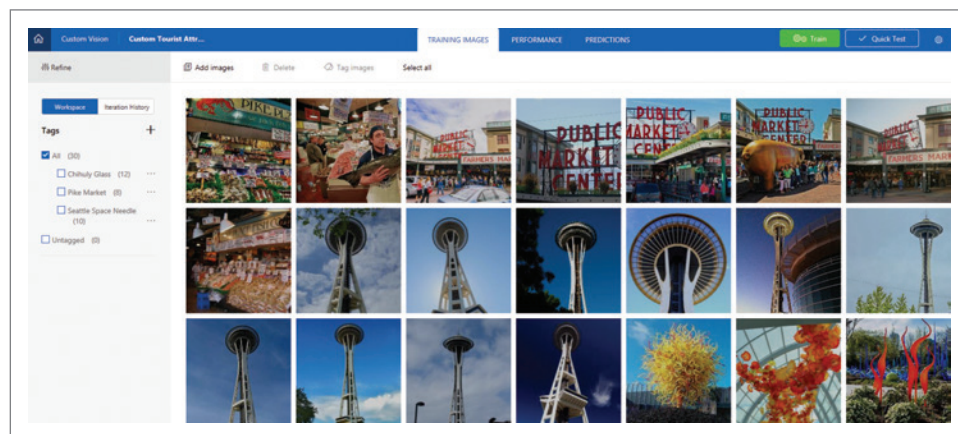


Figure 2 Using CustomVision.AI to Build a Custom Travel Attractions Computer Vision Model

Microsoft AI Platform

The Microsoft AI platform provides a suite of powerful tools, such as the Bot Framework, Cognitive Services, Azure Machine Learning and many more. These tools allow developers to easily and quickly infuse AI into their applications and scenarios, enabling new, intelligent experiences for their users.

Powered by the enterprise-ready capabilities of Azure, the Microsoft AI platform presents a rich set of

Customized Computer Vision

Models: As you explore Cognitive Services to develop your AI applications, you may find that you need to further customize the models using your own data. You can do that with Custom Vision services (customvision.ai). Custom Vision lets you bring your own data, and use it to train your computer vision models. Underneath the hood, state-of-the-art transfer learning techniques leverage existing pre-trained computer vision models, and evolve them to learn about the new images you



Figure 3 Realtime Crowd Insights Solution with Cognitive Services

upload to the Custom Vision service, as depicted in **Figure 2**. This state-of-the-art system allows you to develop a highly performing computer vision model in just minutes.

Custom Machine Learning and Deep Learning Models:

As you work on various use cases, data scientists in your organization might need to develop and customize deep learning models, using various deep learning toolkits. The Microsoft AI platform provides an open and flexible environment for that deep learning. Azure Machine Learning empowers data scientists to build, develop and manage models at scale, while data stores like CosmosDB, SQL DB, SQL Data Warehouse (DW) and Azure Data Lake (ADL) provide access to the structured and unstructured data that inform your ML and deep learning models.

With Azure Machine Learning, you can easily train your models in Spark, run them on Azure Deep Learning Virtual Machines (DLVM), or process them on a managed GPU cluster with Batch AI, and more. Azure Machine Learning experimentation and model services boost productivity by helping you keep track of your projects, enabling you to train on both local and remote compute infrastructures, create containers for model deployment, and manage and monitor the behavior of models.

Bots provide exciting new ways to engage with customers and employees, helping them complete tasks. The Bot Framework provides a rich set of capabilities for conversational AI, so you can develop powerful new bots that interact with your customers and employ-

ees via Web sites, applications, text/SMS, Skype and more.

Tools like Visual Studio Tools for AI, Azure Machine Learning Studio and Azure Machine Learning Workbench provide a great starting point to get started building innovative, intelligent AI applications.

Let's get started with Microsoft AI by using the various services to build an AI application that leverages the intelligent cloud and can be deployed to the intelligent edge. I'll start with Cognitive Services, then move on to building custom models with Azure Machine Learning. I'll finish with a dive into the Bot Framework and show how you can turn any bot into an intelligent bot powered by Microsoft AI.

Cognitive Services

You can add AI capabilities to any .NET app you're developing using Cognitive Services. Let's get started with the Intelligent Kiosk sample app found on the .NET Machine Learning and AI Web site at bit.ly/2yNtRpF. You can access the source code from bit.ly/2zysSKJ.

The Intelligent Kiosk sample app shows how to use Cognitive Services in different scenarios, including:

- Customizing product recommendations based on detected gender and age of visitors.
- Building a real-time AI pipeline to analyze visitor demographics for retail.
- Performing automatic photo capture and face identification.

Let's consider the scenario of building a real-time AI pipeline to analyze the demographics of people visiting a retail store called Realtime Crowd Insights. In this scenario, the Realtime Crowd Insights app continuously analyzes key frames from the live video

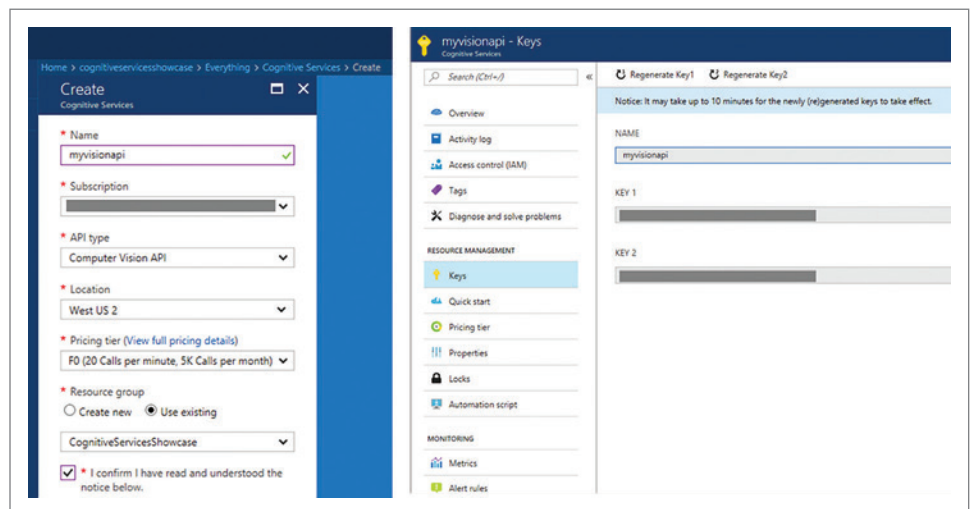


Figure 4 Creating a Cognitive Service and Obtaining the API Key Using the Azure Portal

We didn't invent the Internet...

...but our components help you power the apps that bring it to business.



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
AS2, AS4, EDI/X12, OFTP ...
- **Credit Card Processing**
Authorize.Net, ACH, 3-D Secure ...
- **Shipping & Tracking**
FedEx, UPS, USPS ...
- **Accounting & Banking**
QuickBooks, OFX, SAP ...
- **Internet Business**
Amazon, PayPal, Google ...
- **Internet Protocols**
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**
SSH, SFTP, SSL, Certificates ...
- **Encryption & Certificates**
X.509, OpenPGP, SHA, S/MIME ...
- **Network Management**
SNMP, MIB, LDAP, Monitoring ...
- **Compression**
Zip, Gzip, Jar, AES, 7Zip ...



Our **Red Carpet Subscription** includes all product lines + updates for one year.

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. For more than 20 years, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

stream from the kiosk camera. It then uses Cognitive Services to assign each visitor an anonymous unique identifier, so it can count the number of unique visitors to the kiosk. The software uses facial recognition to determine the age and gender of visitors, as well as recognize displayed emotion. **Figure 3** shows the solution in action.

Azure Machine Learning services
is an integrated, end-to-end
data science and advanced
analytics solution for professional
data scientists to prepare data,
develop experiments and deploy
models at cloud scale.

Once you've downloaded the sample code, you'll learn how to extract key frames from the camera at one frame per second (fps), sending each frame to the Cognitive Services Computer Vision APIs to identify the age, gender and emotion of people present in each frame in the file `RealTimeDemo.xaml.cs` (bit.ly/2hZZDFF). Lines 160 and 161 of this file show how you can invoke the methods for

emotion and face identification (specifically, `DetectEmotionAsync` and `DetectFacesAsync`). Various face attributes, such as bounding box, age and gender are returned by Cognitive Services.

As you explore the code, you'll find the image analysis methods in the file `ImageAnalyzer.cs`. `ImageAnalyzer` further leverages helper classes (like `FaceServiceHelper` and `EmotionServiceHelper`) to leverage Cognitive Services for face and emotion detection. In `FaceServiceHelper`, you'll see how you can use the `FaceServiceClient` provided by Cognitive Services to instantiate the `faceClient` that's used for face detection.

```
private static void InitializeFaceServiceClient() {  
    faceClient = ApiKeyRegion != null ?  
        new FaceServiceClient(ApiKey,  
            string.Format("https://{0}.api.cognitive.microsoft.com/face/v1.0",  
                ApiKeyRegion)) : new FaceServiceClient(ApiKey);  
}
```

You'll find that this is a common pattern used when working with Cognitive Services. Each of the Cognitive Services provides REST APIs, as well as an SDK that makes it easy to use Cognitive Services from any application. The SDK lets you quickly get started developing AI applications using C#, Java, JavaScript, PHP, Python and Ruby. You can also use Curl to directly access the Cognitive Services APIs. Learn more about the APIs at aka.ms/msdn/cognitiveservices/restapi and the Cognitive Services SDK at aka.ms/msdn/cognitiveservices/sdk.

The API key for Cognitive Services can be obtained from the Azure portal after the Cognitive Service you need has been created. When you create a Cognitive Service using the Azure portal, you need to specify the API type that you require—for example, Computer Vision. When developing applications, use the API Key to instantiate each service client.

Figure 4 shows how you can create a Computer Vision Cognitive Service using the Azure portal, and then obtain the API key.

Inside Azure Machine Learning

Azure Machine Learning services is an integrated, end-to-end data science and advanced analytics solution for professional data scientists to prepare data, develop experiments and deploy models at cloud scale. The package enables data science teams to have an environment that enables them to be productive, and amplifies the data science work that they do each day.

When developing custom machine learning and deep learning models, there are lots of choices in the toolkits. Azure Machine Learning lets you use the toolkits you're familiar with—Cognitive Toolkit (CNTK), Tensorflow, Caffe and more. In addition, Azure Machine Learning can deploy, manage and

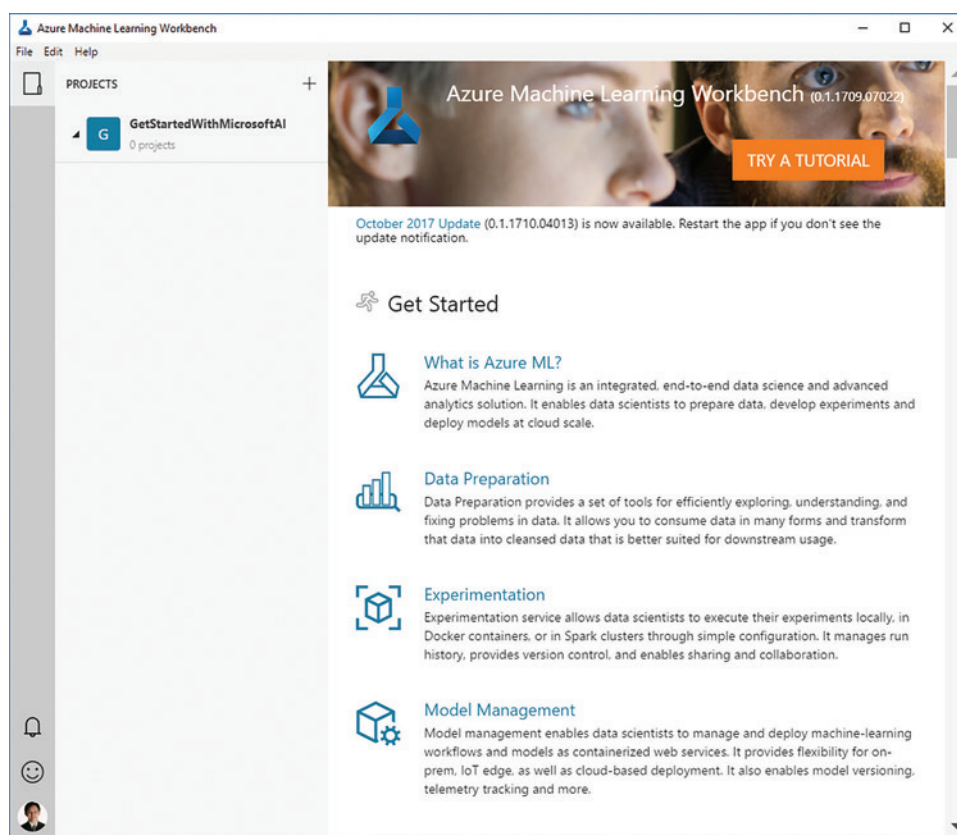


Figure 5 Azure Machine Learning Workbench

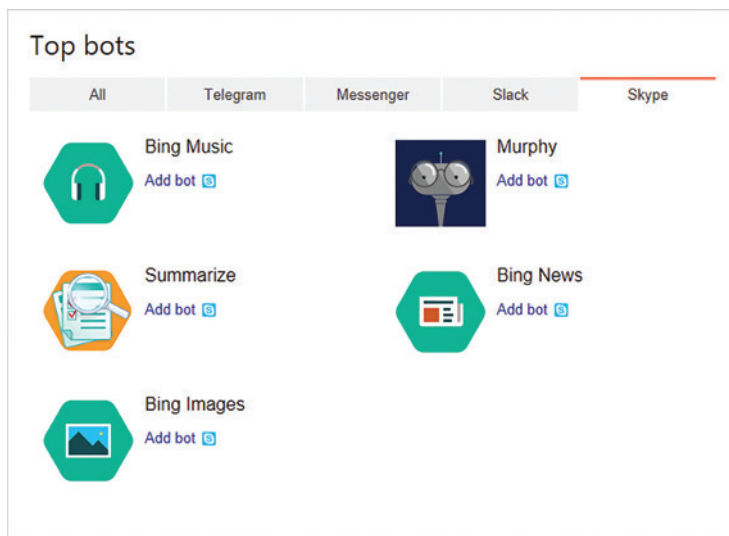


Figure 6 Top Bots Being Used in Skype

monitor your models at scale. Whether deploying to the cloud, to a data lake, to a database like SQL Server 2017 or to the intelligent edge, Azure Machine Learning provides outstanding flexibility for model deployment.

Azure Machine Learning lets you use the toolkits you're familiar with—Cognitive Toolkit (CNTK), Tensorflow, Caffe and more.

The main components of Azure Machine Learning are:

- **Azure Machine Learning Workbench:** Consists of a desktop application and command-line tools that let you manage the entire data science lifecycle, from data ingestion and preparation to model development and deployment. Machine Learning Workbench is available on both Windows and macOS. **Figure 5** shows the Machine Learning Workbench UI.
- **Experimentation Service:** Runs training models across different machine learning environments, ranging from a local machine to a Docker container on a remote virtual machine (VM) to a scaled-out Spark cluster on Azure in the cloud. Experimentation Service integrates with Machine Learning Workbench and supports Git integration, access control and sharing among co-workers.
- **Model Management Service:** Enables management and deployment of machine learning workflows and models. The Model Management Service keeps track of different model versions, and can package and deploy machine learning models as REST APIs served from a Docker container.

Once you've created the Azure Machine Learning Experimentation and Model Management Service, you can download the Machine Learning Workbench to help with your data preparation tasks, as well as manage all your machine learning projects.

Building Bots

Companies have started using intelligent bots to enable users to interact with their services. Bots provide tremendous value in a host of scenarios, including customer support engagements, answering product-related questions, providing tourists with navigation instructions or responding to HR-related questions. We see bots being used in exciting new ways and in different industries, including retail, health care, manufacturing, telecommunications and the public sector.

For example, the AzureBot lets you manage your Azure services (for instance, starting or stopping Azure VMs) using a bot that's available on Skype, Microsoft Teams and other platforms. The Summarize bot uses Bing to present the main points from any Web page. **Figure 6** shows some of the top bots currently being used in Skype (bing.com/search?q=top+bots).

You can build an intelligent bot in minutes using the Azure Bot Service, then deploy it to reach customers on multiple channels, like Skype, Messenger, Microsoft Teams or via a Web site.

To jumpstart development of your first bot, leverage the QnA Maker, a service that makes it easy to create a question and answer bot from an existing frequently asked questions (FAQ) page. Underneath the hood, QnA maker uses state-of-the-art machine learning algorithms and natural language processing (NLP) to distill the information found on FAQ pages to create question-and-answer pairs, as depicted in **Figure 7**. Visit qnamaker.ai to find out more.

Let's look at the architecture for building an intelligent bot that's infused with a range of AI capabilities. These include Cognitive

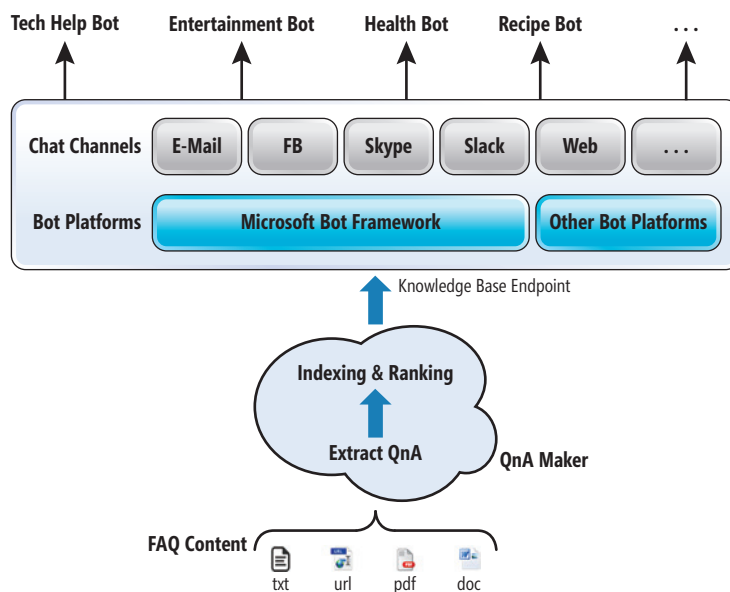


Figure 7 QnA Maker Architecture

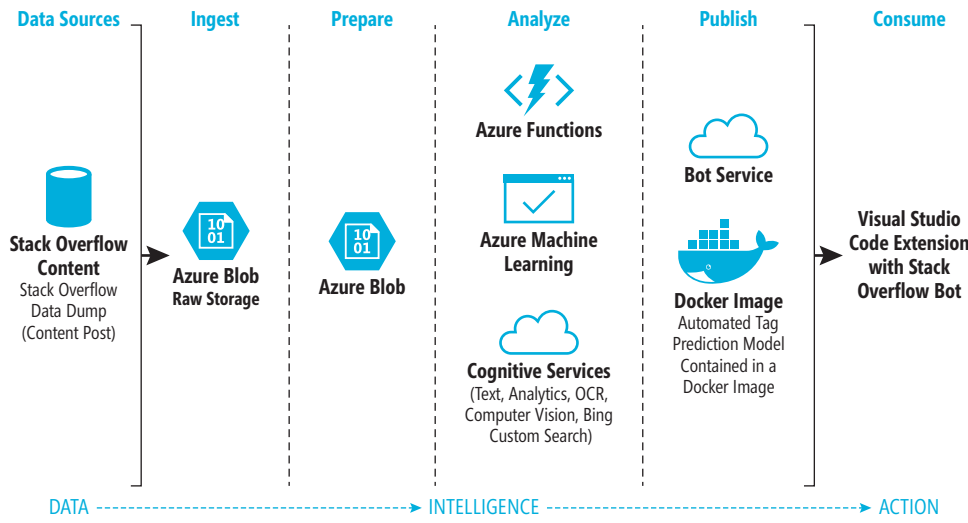


Figure 8 Stack Overflow Bot Architecture

Services like OCR, Text Analytics, Computer Vision and the Language Understanding Intelligent Service (LUIS), as well as program synthesis capabilities from the PROSE SDK (microsoft.github.io/prose) that enables developers to add intelligence to bots. These services are used to power the Stack Overflow bot, which enables developers to get answers to questions within their development environment. You can see the Stack Overflow bot in action at bit.ly/2hqNtEst, and access the sample code at bit.ly/2ziDbZ9.

The Stack Overflow bot leverages Azure Functions to power its dialog analysis. An intelligent Azure Function orchestrates the components used to analyze a user-uploaded screenshot, calling on OCR to extract text in the image and Text Analytics to identify key phrases. **Figure 8** shows the Stack Overflow Bot architecture.

Resources

You can learn more about what's new with Azure Machine Learning at bit.ly/2zVdpUs. Also, there are lots of resources available to help you get started with Azure Machine Learning. Kick things off with the setup and installation guide at aka.ms/aml-blog-setup and a quick start sample from aka.ms/aml-blog-iris. As you explore Azure Machine Learning dive deeper into more in-depth tutorials. Check out a three-part Iris classification tutorial at bit.ly/2AwjSSN, as well as a detailed data wrangling tutorial at bit.ly/2yOrJxQ.

In addition, you can find sample code and detailed scenario walk-throughs on several very interesting use cases. These include:

- **Aerial Image Classification:** Distributed training and operationalization of a land-usage classification model (bit.ly/2jf7y5o).
- **Document Collection Analysis:** Developing a robust model for text analytics (bit.ly/2ypziXr).
- **Predictive Maintenance:** Building an end-to-end predictive maintenance solution using PySpark (bit.ly/2zyBWON).
- **Energy Demand Time Series Forecasting:** Forecasting energy demands to predict future loads on an electrical grid (bit.ly/2hps9mL).

The possibilities of using AI to solve exciting real-world problems are endless! I can't wait to see how you use the Microsoft AI platform to create the next breakthrough intelligent solution.

Program Synthesis by Example (PROSE) enables the bot to generate code by learning from examples provided by the user. This lets developers quickly use generated code to perform transformations of input and output strings. For example, the developer provides an input string like "Jane Doe" and specifies the output string as "Doe, J.". PROSE learns from this and generates the code that then performs the transformation.

In addition, data scientists can build custom deep learning models using Azure Machine Learning to automatically predict which Stack Overflow tags will be most relevant to the questions asked. This

enables the Stack Overflow bot to expand the keywords used to perform keyword searches with Bing Search, helping developers find the answer they need faster.

Wrapping Up

In this article, you learned about the exciting capabilities offered by the Microsoft AI platform. From Cognitive Services that enable you to jumpstart using AI for building intelligent applications, to customizing state-of-the-state computer vision deep learning models, to building deep learning models of your own with Azure Machine Learning, the Microsoft AI platform equips developers with the tools they need. The AI platform is open and flexible, and empowers developers to choose the technology and deep learning framework best suited for their scenarios and skills.

This is the beginning of an intelligent revolution, and the Microsoft AI platform empowers you with exciting enterprise-ready services, infrastructure and tools to build intelligent, innovative applications. Get started today with microsoft.com/ai! ■

JOSEPH SIROSH is the corporate vice president of the Cloud AI Platform at Microsoft, where he leads the company's enterprise AI strategy and products such as Azure Machine Learning, Azure Cognitive Services, Azure Search and the Bot Framework. He's passionate about machine learning and its applications and has been active in the field since 1990. Sirosh holds a doctorate in computer science from the University of Texas at Austin and a BTech in computer science and engineering from the Indian Institute of Technology Chennai.

WEE HYONG TOK is a principal data science manager of the Cloud AI Platform at Microsoft, where he leads AI Prototyping and Innovation. He has advised many Fortune 500 companies on data platform architectures, and using AI for their strategic initiatives. Wee Hyong holds a doctorate in computer science from the National University of Singapore.

THANKS to the following Microsoft technical expert for reviewing this article: Anand Raman, chief of staff of the Cloud AI Platform at Microsoft



DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.



Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

All trademarks or registered trademarks are property of their respective owners.

Delivering On-Device Machine Learning Solutions

Larry O'Brien

You've read the headlines: Artificial intelligence and machine learning (AI/ML) technologies are rewriting the benchmarks across a vast swath of hard problems. Whether it's AlphaGo besting the best human Go player or AlphaGo Zero beating *that* in three days of learning the game from scratch, or Microsoft Research setting a new benchmark for conversation speech recognition, every week seems to bring some new advance built on "deep learning" and "artificial neural networks." Or perhaps you've been more interested in the headlines about bidding wars on the salaries and signing bonuses for developers with ML competence. Either way, the AI/ML train is leaving the station and you don't want to be left behind.

While AI/ML research is advancing at a truly giddy pace, a less celebrated but equally exciting trend is the availability of easy-to-use

libraries for delivering ML functionality on mobile and edge devices. CoreML on iOS and Tensorflow Android Inference on Android are straightforward and consistent once you understand the tools and workflow. As a career strategy, competence in ML technologies is one of the hottest ways toward career flexibility and higher compensation. (From Twitter: "It's AI when you're raising money, it's ML when you're hiring devs.")

It's easy to be intimidated by the latest headlines about AI systems achieving superhuman performance in voice transcription or game-playing, but as Satya Nadella writes in his book, "Hit Refresh" (HarperBusiness, 2017), "Every organization today needs new cloud-based infrastructure and applications that can convert vast amounts of data into predictive and analytical power through the use of advanced analytics, machine learning, and AI."

Many articles and demos about device-based ML focus on vision tasks. This is an area where there has been truly astounding advancement in the past decade. Object detection, captioning and image-to-image style transfer have all advanced at a blistering pace. Azure Cognitive Services CustomVision (customvision.ai) makes it ridiculously simple to develop custom classification models that can be deployed on iOS using the techniques described in this article.

While visual and audio understanding are both inherently important and historically challenging, the "deep learning" revolution in ML goes well beyond these areas. Pattern recognition is at the core of modern ML. Many developers work in areas where recognizing patterns in complex and noisy data is central to their

This article discusses:

- Developing a machine learning model using Keras and Tensorflow
- Converting models to CoreML and Tensorflow protobuf files
- Loading and inferencing using CoreML on iOS and Tensorflow Android Inference on Android
- Using Xamarin.Forms to create a common UX and UI for inferencing

Technologies discussed:

Keras, Tensorflow, Pandas, Long Short Term Memory, Deep Neural Networks, CoreML, Tensorflow Android Inference, Xamarin.Forms, Python, C#

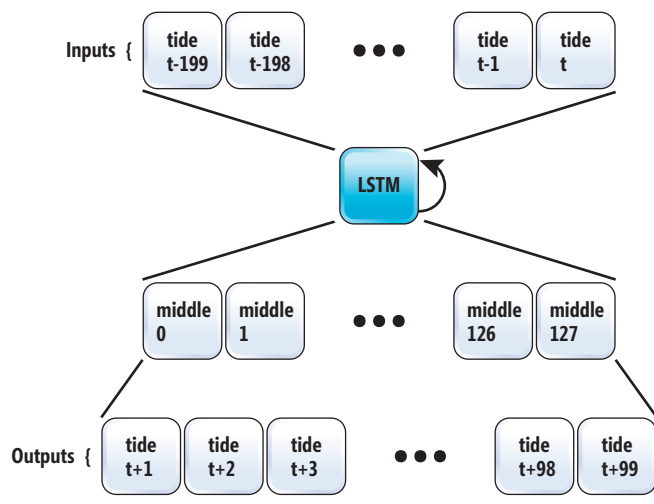


Figure 1 Schematic of the Tide-Prediction Neural Network

business' value proposition. Classifying data, time-series projection, sequence modeling and even sequence-to-sequence construction are all areas where modern ML may lead to competitive advantage. Many developers, for instance, face the problem of "time-series prediction," in which they must reason from large amounts of data that have some structure but are very noisy or have many factors contributing to the ups and downs.

A historically important time-series prediction problem is tide prediction. Predicting the tides was, of course, crucially important in the Ages of Sail and Steam both for merchants and the military. One of the most important artifacts of the early days of computing is Lord Kelvin's tide-predicting machine, described in Charles Petzold's work-in-progress "Computer of the Tides" (bit.ly/2yEhaxk), as "a magnificent assemblage of brass and wood that stands as tall as a person, as gorgeous as it is mysterious." The timing of tides is primarily dictated by the geometry of the earth, moon, and sun and by the complex flooding and draining of bays. The forces are so complex that modern accurate tide prediction uses more than 30 site-specific harmonic components, whose values are derived from hundreds of tide gauges spread across the globe. A complete cycle of the system takes 19 years.

Predicting tides is a reasonably difficult task for a modern ML approach. Given 200 historical readings of water level taken every three hours, how accurately can the tide be predicted up to 300 hours into the future?

Modeling the Problem

As chance would have it, I have several thousands of lines of tide-predicting F# code in one of my "finish someday" projects, and it was trivial for me to generate data based on a real harbor (which I'll call "Contoso Harbor" so that no one is tempted to use this code for navigation). To make the task both more difficult and more real-to-life, I added random noise to the training and validation sets (normally distributed, with a standard deviation of 1.5").

There are many deep-learning libraries available to developers. The nitty-gritty of deep learning involves lots of parallel multiplication and sums over very large arrays of floating-point numbers.

GPU support noticeably speeds up even trivial ML projects, and low-level performance is an area where the various large projects compete with each other, just as with graphic shaders. (Interestingly, ML doesn't generally require high precision, and the emerging field of Tensor Processing Units [TPUs] will probably trade-off increased parallelism and power efficiency for word size.)

However, given an efficient low-level foundation, most non-research-level ML architectures can be described using much higher-level abstractions. Some libraries, such as Keras, provide those abstractions on top of various low-level libraries; other libraries, such as Microsoft's Cognitive Toolkit, provide both high-level and low-level abstractions.

While there are several interchange formats striving to gain mindshare, at the moment there's considerable lock-in to the library you choose for training. If you train in Tensorflow, you most likely have to inference in Tensorflow, if you train in Caffe, you most likely have to inference in Caffe.

Classical neural networks do not have any "memory" of their previous inputs and outputs. This is a serious short-coming when it comes to time-series prediction! Recurrent Neural Networks (RNNs), as their name implies, combine their current input with previous results that are looped back as additional inputs. This allows RNNs to recognize patterns in sequential data. The Long Short Term Memory (LSTM) cell, developed by Sepp Hochreiter and Jürgen Schmidhuber in 1997, is a form of RNN that uses internal "gates" to selectively amplify (remember) or damp (forget) these recurrent connections. Although LSTMs are somewhat old fashioned and do not train as fast as modern variants, they have the distinct advantage of being widely supported. An LSTM cell is at the core of my model.

Classifying data, time-series projection, sequence modeling and even sequence-to-sequence construction are all areas where modern ML may lead to competitive advantage.

Because Tensorflow is necessary for deployment on Android, I chose Keras on top of Tensorflow to develop the model for this project. The Keras high-level description of my model looks like this:

```
def build_model(lookback_length, input_feature_count, lstm_layer_output_dimensions,
                prediction_length, dropout_pct):
    model = Sequential()
    model.add(LSTM(lstm_layer_output_dimensions, input_shape=(lookback_length,
                                                                input_feature_count)))
    model.add(Dropout(dropout_pct))
    model.add(Dense(prediction_length))
    model.add(Activation('linear'))
    model.compile(loss='mse', optimizer='rmsprop')
    return model
```

Figure 2 Data Initialization and Model Training

```
data_frame = pd.read_csv("contoso_noisy.txt", names = ["level"])

input_count = 200 # How far to look back
output_count = 100 # How many steps forward to predict
lstm_layer_output_dimensions = 128 # Size of LSTM output
dropout_pct = 0.15 # Dropout density to avoid over-fitting

(training_inputs, training_targets, test_input) =
    dataframe_to_matrices(data_frame, input_count, output_count)
# How many input features? In this case, 1, but changes from model-to-model
features = training_inputs.shape[2]
model = build_model(input_count, features, lstm_layer_output_dimensions,
    output_count, dropout_pct)

# Train (Experimentally, ~0.12 seems to be an "elbow" --
    lower ThresholdStop to gain accuracy by spending training time)
training_history = model.fit(training_inputs, training_targets,
    epochs=2500, batch_size=100, validation_split=0.15,
    callbacks=[ThresholdStop(0.12)])

# Predict and output results, using input data held back from training
predicted = model.predict(test_input)
```

The Long Short Term Memory (LSTM) cell looks back look-back_length samples at an input that has input_feature_count features. In this case, I have only one input feature: the input water levels at previous three-hour intervals. The output of the LSTM layer feeds into a densely interconnected layer that maps from an array of size lstm_layer_output_dimensions to an array of prediction_length that contains the model's predictions of the water level at future intervals. **Figure 1** shows a schematic of the architecture.

This is about as plain-vanilla a model as one could imagine for a time-series prediction problem. The LSTM cell is a kind of RNN.

The nitty-gritty of deep learning involves lots of parallel multiplication and sums over very large arrays of floating-point numbers.

Figure 2 shows how the model is built and trained. I use Pandas to read the training and validation data from the file contoso_noisy.txt and set the constants for a particular training experiment—in this case, looking back 200 steps, looking forward 100—with a 128-element hidden layer. The dropout_density sets a random percentage of input data to zero during training, which is immensely helpful for avoiding over-fitting (the problem of the model learning the specific training data and not generalizing to new situations). I convert the input data_frame to inputs and outputs for training and testing (the data-munging function dataframe_to_matrices isn't shown, but is available in the source code distribution). I call the previously discussed build_model function and then call the model.fit function. This hours-long call adjusts the model's internal values every 100 passes, and repeats either 2,500 times or once the error of the model drops below 12 percent of 1 foot, holding back 15 percent of the data for the validation step.

Figure 3 A Typical Training Session Begins

```
>python Train.py
Using TensorFlow backend.
Train on 2295 samples, validate on 405 samples
Epoch 1/2500
2017-10-30 21:51:49.576493: W c:\tf_jenkins\home\workspace\release-win\m\
windows-gpu\py\35\tensorflow\core\platform\cpu_feature_guard.cc:45] The
TensorFlow library wasn't compiled to use FMA instructions, but these are
available on your machine and could speed up CPU computations.
2017-10-30 21:51:50.155264: I c:\tf_jenkins\home\workspace\release-win\m\
windows-gpu\py\35\tensorflow\core\common_runtime\gpu\gpu_device.cc:940]
Found device 0 with properties:
name: GeForce GTX 960M
major: 5 minor: 0 memoryClockRate (GHz) 1.0975
pciBusID 0000:01:00:0
Total memory: 2.00GiB
Free memory: 1.65GiB
2017-10-30 21:51:50.166001: I c:\tf_jenkins\home\workspace\release-win\m\
windows-gpu\py\35\tensorflow\core\common_runtime\gpu\gpu_device.cc:1030]
Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX
960M, pci bus id: 0000:01:00:0)
2295/2295 [=====] - 9s - loss: 4.4804 - val_loss: 2.8267
Epoch 2/2500
2295/2295 [=====] - 6s - loss: 3.0078 - val_loss: 2.8101
Epoch 3/2500
2295/2295 [=====] - 6s - loss: 2.8734 - val_loss: 2.6333
Epoch 4/2500
2295/2295 [=====] - 6s - loss: 2.5907 - val_loss: 2.2159
Epoch 5/2500
2295/2295 [=====] - 6s - loss: 1.8314 - val_loss: 1.1734
Epoch 6/2500
2295/2295 [=====] - 6s - loss: 0.9937 - val_loss: 0.7333
Epoch 7/2500
2295/2295 [=====] - 6s - loss: 0.7608 - val_loss: 0.6626
Epoch 8/2500
2295/2295 [=====] - 6s - loss: 0.6948 - val_loss: 0.6373
...
```

The first few epochs of a typical run are shown in **Figure 3** and training and validation errors are shown in **Figure 4**.

Experienced ML developers might raise their eyebrows at the curves in **Figure 4**, which show the validation error (the test of the model against data held back from training) being less than that on the training data for quite a while; but the Loss curve includes that data held back by dropout, which artificially raises the error. The graph truncates the Y axis and so it doesn't show the error for the first few dozen epochs. In general, it's a pretty good curve, with no sign of overfitting, and pretty rapid convergence on an error around .15 percent of a foot, which is a little less than 2 inches. Pretty good for a noisy training set!

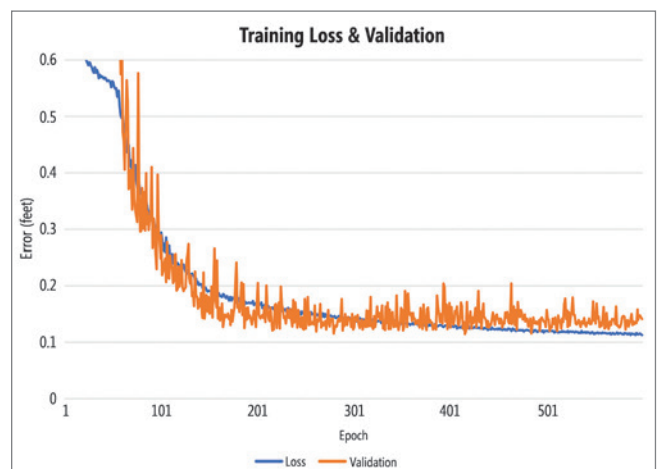


Figure 4 A Typical Training Run



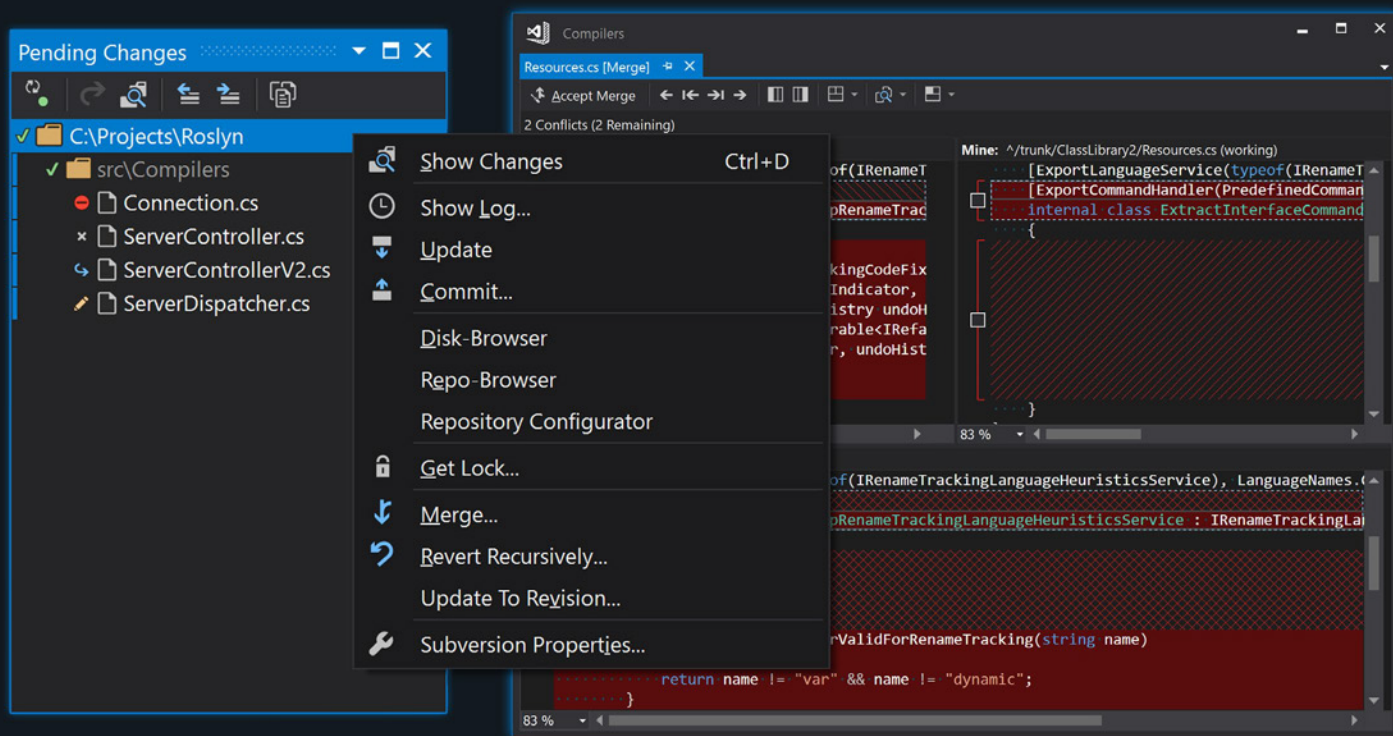
Five-time Award Winner
by Visual Studio Magazine
Reader's Choice Award

VisualSVN: Painless Version Control for Visual Studio

There is something in VisualSVN that makes professional developers love it.

We believe it is a fine-grained mix of a perfect fit with the Apache Subversion® workflow, high performance to support large projects, strong reliability to never crash or hang the IDE, beautiful look and effortless user experience. A number of unique usability features and excellent technical support make VisualSVN the favorite tool for thousands of software developers.

You might be surprised and we know why. If you're still there, give VisualSVN a try.



To learn more, please visit our website →

www.visualsvn.com

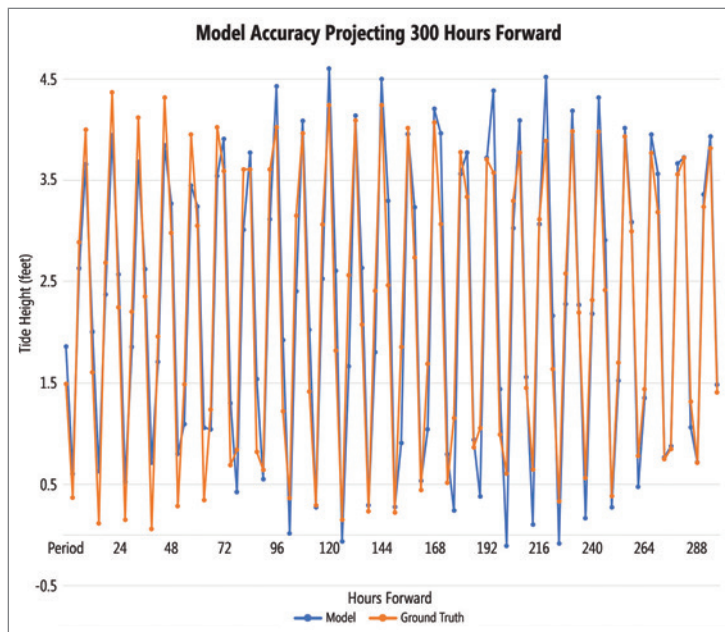


Figure 5 The Neural Net Captures Tide Cycles

Even better, **Figure 5** shows the model and ground-truth predictions going forward 100 timesteps, which for the three-hour timesteps translates into 12 1/2 days. While the amplitudes are off, the timing and general waxing and waning of spring and neap tides are clearly captured by the model. (It's typical that the water level doesn't average to zero over such a short time period.)

The trained model can be saved in the preferred Keras HDF5 format and reloaded and used as necessary to predict tides in Contoso Harbor given recent or historical tide gauge readings.

Conversion and Deployment to Mobile Devices

While many ML scenarios can use a cloud-based Web service for the calculation, there are several reasons why on-device inferencing might be preferable.

Most non-research-level ML architectures can be described using higher-level abstractions.

First and foremost is performance. While mobile devices pale in horsepower compared to the GPUs on Azure N-Series machines, inference is vastly less costly than training. The latest iPhones have the A11 Bionic chip, with hardware dedicated to neural net operations, and the Pixel 2 Pixel Visual Core points the way to similar accelerated capabilities on Android.

While my experience is that on-device inference with typical hardware can take upward of a second with large models, good async programming practices can lead to apps with excellent responsiveness. See, for instance, the CoreMLAzureModel and CoreMLVision samples at bit.ly/2zqQBso, and bit.ly/2AslxJc, both of which perform inference on video streams. The StopWatch class

can be invaluable for understanding the computational cost of your on-device inferencing.

Second, data volumes can be significant. In scenarios that involve continuous inferencing, audio and image data (much less video streams) can chew up bandwidth quickly.

Finally, there will always be the possibility that users just plain don't have Internet access at the moment.

On-device inferencing was introduced in the CoreML framework in iOS 11 and macOS, while Android users can use Tensorflow Android Inference (TAI). In the future, Google's just-announced Neural Networks API (bit.ly/2Aq2fnN), will likely be preferred over this library.

Whether targeting CoreML or TAI, you have to convert the Keras HDF5 file to compatible formats. Conversion to CoreML is simple:

```
import coremltools

# Convert to CoreML
coreml_model = coremltools.converters.keras.convert(
    "keras_model_lstm.hdf5", ["readings"], ["predicted_tide_ft"])
coreml_model.author = 'Larry O'Brien'
coreml_model.license = 'MIT'
coreml_model.save('LSTM_TidePrediction.mlmodel')
```

Figure 6 Extracting and saving Underlying Tensorflow Data from a Keras Model

```
# Derived from code by Amir H. Abdi released under the MIT Public License
# https://github.com/amir-abdi/keras_to_tensorflow/blob/master/keras_to_tensorflow.ipynb

input_fld = '.'
weight_file = 'keras_model_lstm.hdf5'
num_output = 1
write_graph_def_ascii_flag = True
prefix_output_node_names_of_final_network = 'output_node'
output_graph_name = 'TF_LSTM_Inference.pb'

from keras.models import load_model
import tensorflow as tf
import os
import os.path as osp
from keras import backend as K

output_fld = input_fld
if not os.path.isdir(output_fld):
    os.mkdir(output_fld)
weight_file_path = osp.join(input_fld, weight_file)

K.set_learning_phase(0)
net_model = load_model(weight_file_path)

pred = [None]*num_output
pred_node_names = [None]*num_output
for i in range(num_output):
    pred_node_names[i] = prefix_output_node_names_of_final_network+str(i)
    pred[i] = tf.identity(net_model.output[i], name=pred_node_names[i])
    print('output nodes names are: ', pred_node_names)

sess = K.get_session()

if write_graph_def_ascii_flag:
    f = 'only_the_graph_def.pb.ascii'
    tf.train.write_graph(sess.graph.as_graph_def(), output_fld, f, as_text=True)
    print('saved the graph definition in ascii format at: ', osp.join(output_fld, f))

from tensorflow.python.framework import graph_util
from tensorflow.python.framework import graph_io
constant_graph = graph_util.convert_variables_to_constants(
    sess, sess.graph.as_graph_def(), pred_node_names)
graph_io.write_graph(constant_graph, output_fld, output_graph_name, as_text=False)
print('saved the constant graph (ready for inference) at: ', osp.join(
    output_fld, output_graph_name))
```

File Format APIs

Open, Create, Convert, Print and Save files from your applications!

Try risk free - 30 day trial

Aspose.Total

Enable your applications to manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats for all major platforms.



Aspose.Words

Create, edit, convert or print Word documents (DOC, DOCX, RTF etc.) in your .NET, Java and Android applications.



Aspose.Cells

Develop high performance .NET, Java and Android applications to Create, Edit or Convert Excel worksheets (XLS, XLSX, ODS etc).



Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS etc.) using our native APIs for .NET, Java and Android platforms.



Aspose.Slides

Create, edit or convert PowerPoint presentations (PPT, PPTX, ODP etc.) in your .NET, Java and Android applications.



Aspose.Email

Create, Edit or Convert Outlook Email file formats (MSG, PST, EML etc.) and popular network protocols.



Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet etc.) using our native APIs for .NET and Java.

► Aspose.Imaging ► Aspose.Tasks ► Aspose.OCR ► Aspose.Diagram ► Aspose.Note ► Aspose.HTML



ASPOSE
File Format APIs

Download a Free Trial at
<https://downloads.aspose.com>

Americas: +1 903 306 1676

EMEA: +44 141 628 8900
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

The CoreML code relies on the `coremltools` package written by Apple, whose source code is available under the 3-Clause BSD License at bit.ly/2m1gn3E. CoreML works with a large number of ML models, including non-neural network models, such as Support Vector Machines, tree ensembles, and linear and logistic regression models. (See the table at the Xamarin's CoreML API documentation homepage, bit.ly/2htyUkc.)

Because this model was trained using Tensorflow, I can extract the underlying Tensorflow computational graph and save the weights, as shown in **Figure 6**. This code is derived from the work of Amir Abdi (bit.ly/2ApB5ND).

Writing the Apps

With a well-performing model in hand and converted to device formats, it's time to develop the app. Although the ML inferencing specifics differ between iOS and Android, naturally I'll have a single source for UI code via Xamarin.Forms.

The completion solution (available at bit.ly/2j7xcsM) contains four projects: the Xamarin.Forms shared-code project that defines the UX, its iOS and Android implementations, and an Android binding project for the TAI library.

Although Tensorflow is associated with Google, shipping versions of Android don't have built-in support for ML models. Instead, this project uses the easy-to-use `TensorFlowInferenceInterface` class, which is defined in the `Org.Tensorflow.Contrib.Android` namespace and distributed in an 11MB shared library. Projects under Tensorflow's `Contrib` directory aren't officially supported, but this project appears to have an active community of committers and I suspect it will continue into the future.

Although binding an Android or iOS native library sometimes has complexities, in this case the binding project is trivial. It's

literally just a matter of putting the Java files in the appropriate place in the `/Jars` subdirectory and letting the Xamarin infrastructure take care of the rest.

The Android project's `/Assets` directory should contain a copy of the Tensorflow protobuf weight file generated by `Keras2Tensorflow.py`. To use the `.mlmodel` file produced by `Keras2CoreML`, though, an additional step is required. While the `.mlmodel` file is good for sharing, actual runtime use requires it to be compiled into a different format.

On-device inferencing is quite simple and consistent between projects.

On a Macintosh that has Xcode installed, you convert a `.mlmodel` using `xcrun compile LSTM_TidePrediction.mlmodel LSTM_TidePrediction.mlmodelc` that contains several model-defining files. This folder should be moved to the iOS project's `Resources` folder. (The source-code distribution has already performed all the steps necessary and the TAI library, Tensorflow and CoreML models are all in their proper locations.)

With the project structure in place, let's briefly discuss the UI. This isn't my strong suit, as **Figure 7** amply demonstrates. The app consists of a scrolling graph of predictions (using Aloïs Deniel's `Microcharts` package available on NuGet or bit.ly/2zrl0Gu), three buttons that allow you to load and predict tides based on any of three datasets, and the prediction values themselves.

You'll notice that while the general shape of the overall prediction is consistent, the fine-grained predictions are often in disagreement by as much as a few inches. There are several possible reasons for this. The output of a deep neural net is the product of thousands of floating-point multiplications and sums; differences in low-level representations could easily accumulate to significant levels. Another possibility is differences between CoreML and Tensorflow in the implementation of the LSTM or feed-forward



Figure 7 Xamarin.Forms Under iOS (left) and Android (right), Using CoreML and Tensorflow Android Inference

8 Key Considerations

Use these eight steps to deliver an AI/ML solution on mobile devices:

1. Choose an ML library compatible with the device OS or systems you're targeting.
2. Develop a data-wrangling training pipeline that allows you to rapidly explore your data and iterate your model.
3. Consider using cloud-based resources for final training.
4. Convert your model to device format.
5. Convert your on-device data to the form expected by the model.
6. Treat the on-device inferencing as a "black box" function call.
7. Validate that the on-device inferencing matches your training results.
8. Consider implementing the ability to download a new model, but be aware of the size implications.

nodes. A third possibility is that the differences are introduced during the conversion of the Keras model to CoreML. One way or the other, the differences highlight the importance of validating the model's behavior *on the device*.

Device-Based Inferencing

Compared to the complexity of developing the model, on-device inferencing is quite simple and consistent between projects:

Load the model into the library's inferencing class. Once on the device, the model is essentially a blackbox, with little support for reading or modifying internal state. The inferencing functions can be expressed as an interface in the Xamarin.Forms base project:

```
public interface ITidePredictor
{
    /// <summary>
    /// From 200 input sea levels, in feet, taken every 3 hours,
    /// predict 100 new sea levels (next 300 hours)
    /// </summary>
    /// <returns>100 levels, in feet, representing the predicted tide
    /// from the final input time to + i * 3 hours</returns>
    /// <param name="sealevelInputs">200 water levels, measured in feet,
    /// taken every 3 hours</param>
    float[] Predict(float[] seaLevelInputs);
}
```

The native Android class for inferencing is `Org.Tensorflow.Contrib.Android.TensorflowInferenceInterface` and the iOS class is `CoreML.MLModel`. Both are loaded in their device project's constructor, as shown in **Figures 8** and **Figure 9**.

In both cases, the model is loaded from a file, but both CoreML and TAI support loading a model from a Web-based URL. While a Web-based model is obviously easier to update, the tradeoff is that many ML models are extremely large. The tide prediction model is only a few hundred kilobytes in size, but image-recognition models often weigh in at hundreds of megabytes.

Configure input data. Because ML libraries have their own internal datatypes and structures, almost all interact with calling programs using dictionaries of strings to input and output data. Although the names can be set in Keras, conversion to CoreML and Tensorflow has a tendency to mangle them. In the case of Android, the input sealevels are associated with the string "lstm_1_input" and

Figure 8 The Android Inferencing Class

```
public class TensorflowInferencePredictor : ITidePredictor
{
    const string MODEL_FILE_URL = "file:///android_asset/TF_LSTM_Inference.pb";
    const string INPUT_ARGUMENT_NAME = "lstm_1_input";
    const string OUTPUT_VARIABLE_NAME = "output_node0";
    const int OUTPUT_SIZE = 100;

    TensorFlowInferenceInterface inferenceInterface;

    public TensorflowInferencePredictor(AssetManager assetManager)
    {
        inferenceInterface = new TensorFlowInferenceInterface(
            assetManager, MODEL_FILE_URL);
    }

    public float[] Predict(float[] inputSeaLevels)
    {
        inferenceInterface.Feed(INPUT_ARGUMENT_NAME, inputSeaLevels,
            inputSeaLevels.Length, 1, 1);
        inferenceInterface.Run(new string[] { OUTPUT_VARIABLE_NAME });
        float[] predictions = new float[OUTPUT_SIZE];
        inferenceInterface.Fetch(OUTPUT_VARIABLE_NAME, predictions);
        return predictions;
    }
}
```

the output predictions with "output_node0." Configuring input is easy in Android, as conversion isn't necessary. As you can see in the call to `Feed` in **Figure 8**, the input array is passed, followed by `inputSeaLevels.Length, 1, 1`. This encodes the shape of the input data: 200 rows, each containing 1 feature defined by 1 value.

CoreML input and output is more complex. While TAI takes and returns managed arrays, CoreML works with `MLFeatureValue` datatypes defined in the CoreML namespace and presumably tuned to Apple hardware. The inputs to the model are defined in the `TideInput` class, shown in **Figure 10**. Note that `TideInput` is defined as implementing the `IMLFeatureProvider` interface. The `MLModel` object knows the names and types of its expected inputs, and uses the `IMLFeatureProvider` interface to retrieve that data. The `FeatureNames` property must mimic the set of expected variables names, and the `GetFeatureValue` method must provide the data for the relevant string.

When converting the Keras tide-prediction model to CoreML, the converter told us that the model takes as input 3 `MLMultiArray` objects. The `TideInput` class needs to initialize those objects. The first is the expected readings input with its [200, 1, 1] shape:

```
var ma = new MLMultiArray(new nint[] { INPUT_SIZE, 1, 1 },
    MLMultiArrayDataType.Double, out m1Err);
for (int i = 0; i < INPUT_SIZE; i++)
{
    ma[i] = tideInputData[i];
}

readings = MLFeatureValue.Create(ma);
```

Figure 9 The iOS Inferencing Class

```
public class CoreMLTidePredictor : NSObject, ITidePredictor
{
    public event EventHandler<EventArgsT<string>> ErrorOccurred = delegate { };

    MLModel model;

    const int OUTPUT_SIZE = 100;
    const int OUTPUT_FIELD_NAME = "predicted_tide_ft";

    public CoreMLTidePredictor()
    {
        // Load the ML model
        var bundle = NSBundle.MainBundle;
        var assetPath = bundle.GetUrlForResource("LSTM_TidePrediction", "mlmodelc");
        NSError m1Err;
        model = MLModel.Create(assetPath, out m1Err);
        if (m1Err != null)
        {
            ErrorOccurred(this, new EventArgsT<string>(m1Err.ToString()));
        }
    }

    public float[] Predict(float[] seaLevelInputs)
    {
        var inputs = new TideInput(seaLevelInputs);
        NSError m1Err;
        var prediction = model.GetPrediction(inputs, out m1Err);
        if (m1Err != null)
        {
            ErrorOccurred(this, new EventArgsT<string>(m1Err.ToString()));
        };

        var predictionMultiArray = prediction.GetFeatureValue(
            OUTPUT_FIELD_NAME).MultiArrayValue;
        var predictedLevels = new float[OUTPUT_SIZE];
        for (int i = 0; i < OUTPUT_SIZE; i++)
        {
            predictedLevels[i] = predictionMultiArray[i].FloatValue;
        }
        return predictedLevels;
    }
}
```

The other expected inputs (lstm_1_h_in and lstm_1_c_in of shape [128, 1, 1]) are more surprising, but “h” is often used for an LSTM’s output and “C” for the cell’s state. Setting all the values for these inputs to 0 results in correct predictions, so that’s what TideInput does.

Call the inferencing function. With everything configured, it’s time for the magic! The Android call `inferenceInterface.Run(new string[] { OUTPUT_VARIABLE_NAME });`, and the iOS call, `var prediction = model.GetPrediction(inputs, out mErr);`, perform the actual inferencing. In both cases, this is a synchronous call. Timing, of course, varies from machine to machine and model to model, and is very quick on the small tide-prediction model. With image-recognition models, though, I’ve seen this function take many hundreds of milliseconds. When writing apps that work with video frames, I’ve used simple background processing and a simple flag to throttle requests to the ML library. In iOS 11.1, Apple added new Pressure Level APIs for its video subsystem that can interrupt capture sessions if the hardware gets too hot, which supports the intuition that things like continuous image recognition and augmented reality are pretty darn processor-intensive!

Figure 10 Configuring CoreML Input

```
class TideInput : NSObject, IMLFeatureProvider
{
    MLFeatureValue readings;
    MLMultiArray lstm_1_h_in, lstm_1_c_in;
    const int INPUT_SIZE = 200;
    const int MIDDLE_SIZE = 128;

    public NSSet<NSString> FeatureNames
    {
        get
        {
            return new NSSet<NSString>(
                new NSString("readings"),
                new NSString("lstm_1_h_in"),
                new NSString("lstm_1_c_in")
            );
        }
    }

    public MLFeatureValue GetFeatureValue(string featureName)
    {
        switch (featureName)
        {
            case "readings": return readings;
            case "lstm_1_h_in": return MLFeatureValue.Create(lstm_1_h_in);
            case "lstm_1_c_in": return MLFeatureValue.Create(lstm_1_c_in);
            default: throw new ArgumentOutOfRangeException();
        }
    }

    public TideInput(float[] tideInputData)
    {
        // 200 elements, 1 batch, 1 feature
        NSError mErr;
        var ma = new MLMultiArray(new nint[] { INPUT_SIZE, 1, 1 },
            MLMultiArrayDataType.Double, out mErr);
        for (int i = 0; i < INPUT_SIZE; i++)
        {
            ma[i] = tideInputData[i];
        }
        readings = MLFeatureValue.Create(ma);
        lstm_1_h_in = new MLMultiArray(new nint[] { MIDDLE_SIZE },
            MLMultiArrayDataType.Double, out mErr);
        lstm_1_c_in = new MLMultiArray(new nint[] { MIDDLE_SIZE },
            MLMultiArrayDataType.Double, out mErr);
        for (int i = 0; i < MIDDLE_SIZE; i++)
        {
            lstm_1_h_in[i] = lstm_1_c_in[i] = new NSNumber(0.0);
        }
    }
}
```

Retrieve the output variable or variables. Just as with the inputs, the outputs of the model are associated with strings. In the case of Android:

```
float[] predictions = new float[OUTPUT_SIZE];
inferenceInterface.Fetch(OUTPUT_VARIABLE_NAME, predictions);
```

These populate the predictions array, while the iOS code is only slightly more complex:

```
var predictionMultiArray =
    prediction.GetFeatureValue(OUTPUT_VARIABLE_NAME).MultiArrayValue;
var predictedLevels = new float[OUTPUT_SIZE];
for (int i = 0; i < OUTPUT_SIZE; i++)
{
    predictedLevels[i] = predictionMultiArray[i].FloatValue;
}
```

Both CoreML and TAI are performance-optimized libraries that do little or no input validation. Data is basically treated as raw C buffers; mistakes in input size, shape or format can result in diagnostic-free crashes or, even more confusingly, complaint-free “Garbage In, Garbage Out” results.

The field of ML is developing at a blistering pace, and it would be a full-time job just to evaluate the latest research papers and industry announcements. While the cutting edge is fascinating, huge amounts of value can be unlocked using techniques and architectures that have been around practically forever (that is, years).

There is a rough progression of difficulty in ML tasks from pattern recognition to classification to sequence modeling to sequence-to-sequence generation. Of course, the signal needs to be present in the data, and recognizing a pattern in a very noisy stream may be more difficult than generating something that acts like a simple Markov model (I’m looking at you, “Neural Net Generates Funny Boat Names!” articles).

Keep It Simple

My ML models are almost exclusively done with high-level abstractions and well-known architectures. Even though I can rarely resist pre-processing my data to emphasize necessary features, this often turns out to be premature optimization. It’s amazing how often “the simplest architecture that could possibly work” manages to extract a signal from unprocessed data. Lord Kelvin’s tide computer relied on the results of a Fourier analysis of water levels, but the simple model I developed reproduces at least the short-term epicycles pretty well. (I plan on seeing if I can generate the classic harmonic components from raw data—stay tuned to my Twitter feed.)

Whether your ambitions involve super-human competence, in-the-field business intelligence, or simply boosting your career prospects, modern ML is a fascinating field with powerful capabilities and seemingly unlimited potential for delivering value. Delivering that ML capability on devices is straightforward with Microsoft’s Xamarin technologies and the techniques described in this article. ■

LARRY O’BRIEN is a senior content publication manager for Microsoft’s Xamarin technologies. His first published technical article was “Developing a Neural Network in C++” for *AI Expert* in 1989. He’s seen ‘em come and he’s seen ‘em go. He’s on Twitter: @lobrien.

THANKS to the following Microsoft technical experts for reviewing this article: Anuj Bhatia and Alexander Kyte



Build Better Applications with LEADTOOLS Imaging SDKs

Developers know the value of a quality toolkit, and LEADTOOLS delivers. With over 27 years of experience, LEAD's comprehensive document (OCR, Barcode, PDF, Forms Recognition (ID, Passport, Invoice), Viewers, Annotations, File Formats), medical (DICOM, PACS, and HL7), and multimedia imaging SDKs. These features, available for .NET, C++, Linux, iOS, macOS, Android, and HTML5/JavaScript, offer developers more than any toolkit on the market.

LEADTOOLS literally puts millions of lines of code at your fingertips and cuts months, if not years off of your development life cycle. Whatever your programming needs, LEADTOOLS provides the best and most diverse imaging technology available.



Download a free, fully functional evaluation SDK →

www.leadtools.com

Introducing App Center

Matt Gibbs

App Center is a collection of services that help you build better apps faster. It brings together the beta distribution and crash analytics features of HockeyApp and the Xamarin Test Cloud ability to run application tests on physical devices. It provides cloud build services so that you can have “continuous integration” running for every code committed to a branch. App Center also provides app usage analytics and the ability to segment users and send them push notifications. You can choose to use just one of the services as part of your app development lifecycle, or combine them into a complete DevOps process.

In this article, I show how to get started with the services of App Center to build, test, distribute, monitor and improve your apps. I'll use an Android app as an example, but you should note that App Center currently supports multiple OSes and development

platforms. The configuration details vary, but the key concepts are the same.

Create an App Center account. Browse to appcenter.ms and select the type of credentials to use. App Center currently supports authenticating with accounts from Microsoft, GitHub, Facebook and Google. Or you can create an account with an e-mail address and App Center password.

Create an App. Click on the New App button and select the Operating System and Platform. OS choices include iOS, Android and Windows. Currently supported development platforms are Xamarin, Java, Objective-C, Swift, React Native and Universal Windows Platform (UWP).

Build

If you're the only dev working on a project, you may just build everything locally. When there is a team of developers working on a project, you might prefer centralized build support. Instead of maintaining your own lab infrastructure, connect App Center to your source code repository and have it do builds for you. Builds can be triggered manually, or you can use Continuous Integration to have App Center build every time new code is pushed to the branches you specify.

Connect to your repo. The navigation sidebar lists the App Center services. Click Build and select the service that hosts your source code. App Center currently supports Git repositories hosted on Visual Studio Team Services (VSTS), Bitbucket and GitHub.

This article discusses:

- Using App Center
- Building an Android app
- Testing in App Center
- Distribution via App Center
- Crash reporting in App Center
- App Center Analytics

Technologies discussed:

App Center, Xamarin, Android, JSON

You'll be prompted to authorize access for App Center. The permissions allow App Center to clone repos and configure webhooks for triggering builds automatically. Select the app repository.

Instead of maintaining
your own lab infrastructure,
connect App Center to your
source code repository and have
it do builds for you.

Configure the build. App Center displays the branches of the repo. Select a branch to configure. App Center inspects the build files to determine what modules and build variants are available. **Figure 1** shows the build configuration options. By default, a build is triggered whenever code is pushed to the branch.

You can trigger a build when saving the configuration to validate that things are set up correctly. The builds for a branch are listed, showing when the build was triggered, the commit messages, a build number and the build status. The status is first set to queued until assigned to a host, when it changes to building. The build will complete with a status of canceled, failed or succeeded.

While the build is running, you can monitor the build output in real time. After the build completes, you have the option to download the logs or the build application package. App Center builds can run unit tests and static code analysis (lint). The build will fail if the unit tests fail or linting reports errors.

Sign the build. Before a build can be installed and run on a device, it must be signed. You can have the keystore in the repo with passwords checked in to gradle files. I don't like or recommend putting credentials in sources, so I uploaded the keystore and credentials separately.

Launch test. App Center build provides the option to test on a real device. This can be enabled without providing your own signing details. The test cloud infrastructure requires that the application have Internet permissions. If you haven't already declared the permission, add the following line to your AndroidManifest.xml:

App Center will sign and deploy the app to a real device and perform a basic launch test to ensure that the application package can be installed and run.

```
<uses-permission android:name="android.permission.INTERNET" />
```

App Center will sign and deploy the app to a real device and perform a basic launch test to ensure that the application package can be installed and run.

Test

Test in the cloud, on real devices. Testing your app locally on an emulator is different than running it on a real device. App Center can deploy your tests, along with your application, to real devices. It captures screenshots at the test steps you define, and provides complete logs of the test run. This gives you access to thousands of device configurations (a specific device model with a specific version of the OS).

Define device sets. Devices in App Center are classified into three tiers according to their popularity. App Center maintains more of the tier 1 devices. Devices are moved from tier 1 to tier 2, and eventually to tier 3. When tests execute, wait times for lower-tier devices may be longer than wait times for tier 1.

Test runs are executed against a collection of devices. In the Test area, click on Device Sets to manage

Figure 1 Build Configuration

your Device Set definitions. **Figure 2** shows the hundreds of Android device configurations available. You can search and filter by form factor, OS version, CPU, device tier, memory, manufacturer and name.

Write tests. The App Center recommendation is to use the testing framework native to the platform. Espresso is the recommended framework for Android. For detailed documentation of Espresso, see bit.ly/2mkW6WZ.

The application and tests need to be updated to run in App Center before you upload them. App Center provides a ReportHelper object that aids in navigating the test report. To make the ReportHelper available at compile time for Espresso tests, include the following dependency in the application gradle file:

```
dependencies {
    androidTestImplementation('com.xamarin.testcloud:espresso-support:1.1')
}
```

Add the Test Cloud imports to your test classes:

```
import com.xamarin.testcloud.espresso.Factory;
import com.xamarin.testcloud.espresso.ReportHelper;
```

Instantiate the ReportHelper in each test class:

```
@Rule
public ReportHelper reportHelper = Factory.getReportHelper();
```

Use the ReportHelper to label the test steps. App Center will annotate the test report with the labels and capture a screenshot of the running app at that point in the test. Use the @After annotation to include a label at the end of the testing. If a test fails, the @After screenshot shows the final state. This can be extremely helpful in figuring out why a test run failed.

```
@After
public void TestComplete() {
    reportHelper.label("Test Complete");
}
```

Compile. To run the tests on devices in the cloud requires uploading the application package and the test package. Create them both with the following commands:

```
gradlew assembleDebug
gradlew assembleDebugAndroidTest
```

Install the CLI. To upload and initiate test runs requires the App Center CLI. To run the CLI, you need at least version 6.3 of Node.js (nodejs.org). Install the CLI using npm:

```
npm install -g appcenter-cli
```

Log in to App Center:

```
appcenter login
```

You will be redirected to the browser to complete the login and get an auth token for the CLI. The CLI is accessing the App Center REST API. The same auth token can be used to access the API directory.

Define a Test run. In the Test area, click on Test runs to see the history and status of tests. A test run is the combination of a test package run on a device collection. You specify a test series label for each test run so that results can be

grouped. You can have test packages focused on different areas of functionality and reuse device collections with different test packages. Click on New Test run and create a new device collection, or select one you've already defined. Specify the test series label, system language of the devices and the test framework. App Center now displays instructions for using the CLI to upload the test files and initiate the test run. Copy the command from the portal and paste it into your shell. Customize the parameters for your local setup.

You can have test packages focused on different areas of functionality and reuse device collections with different test packages.

Here's the command for uploading my locally built packages and running the tests against my latest-Android device set. The results are part of the main-activity test series:

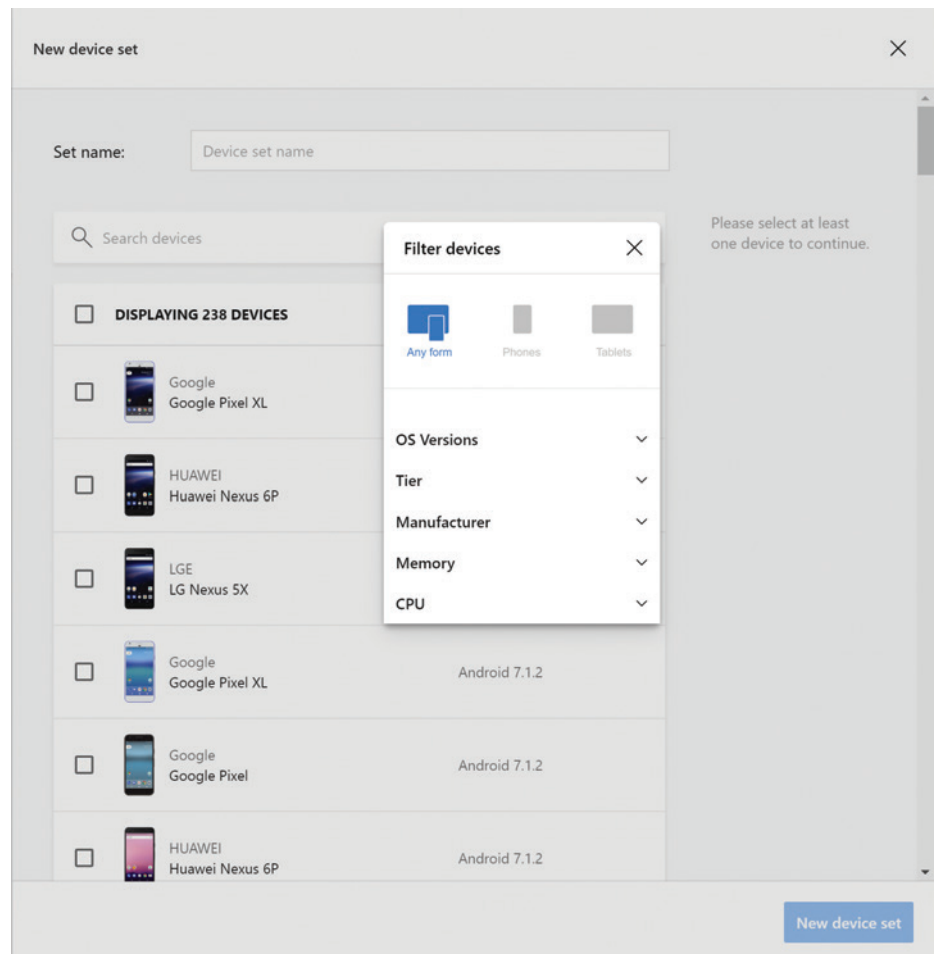


Figure 2 New Device Set



Thank You x 19

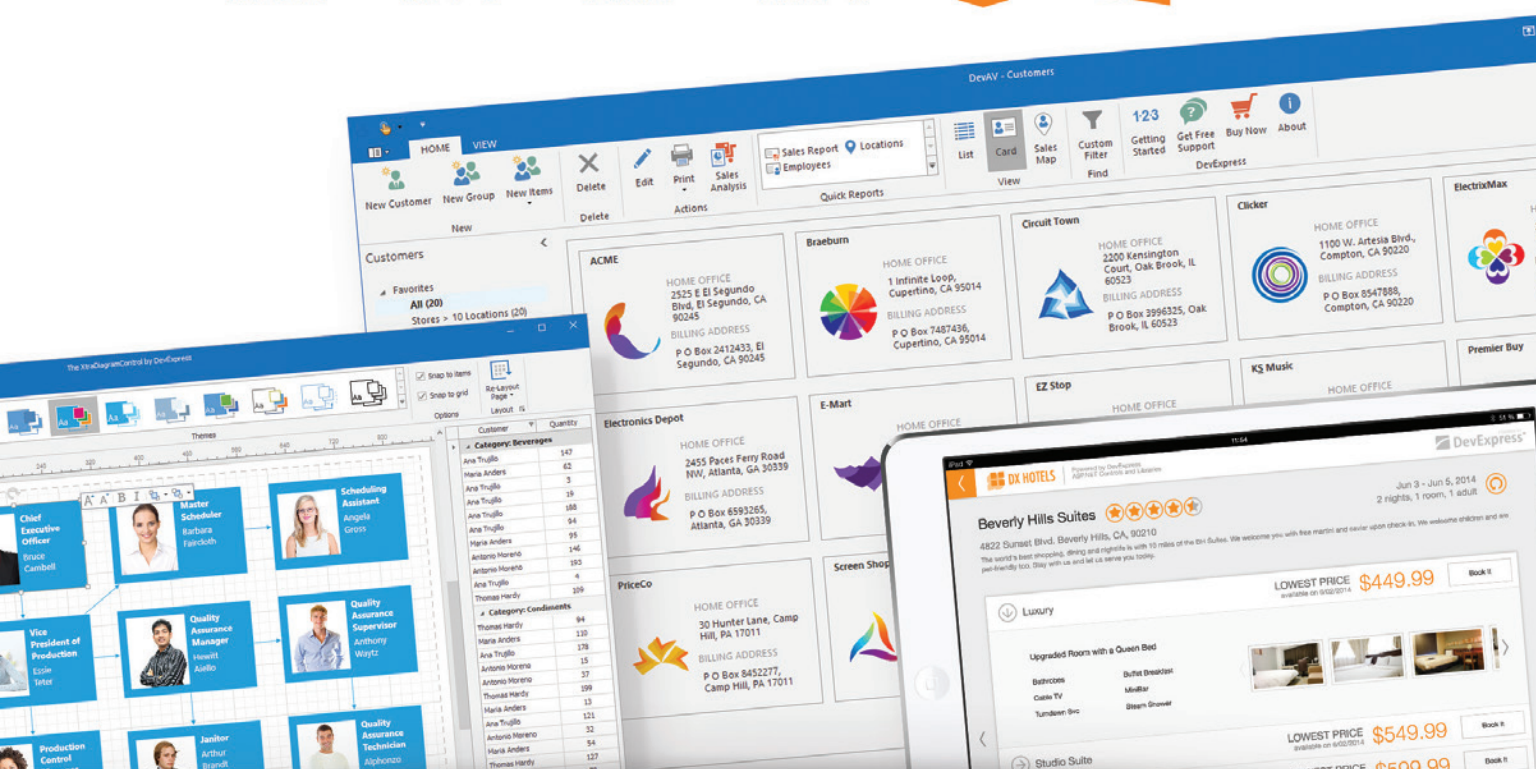
We are grateful to everyone who voted for our products in this year's VSM Readers' Choice Awards. Thank you for your continued support and for the confidence you've placed in DevExpress over the last 19 years.

Since 1998, our goal has been steadfast and unshakeable: Create best-of-breed software development tools so you can deliver high-impact business solutions that amaze. From the desktop and web to your mobile world, we remain fully committed to your needs and will do everything possible to earn your trust in the years to come.

Thank you once again from all of us at DevExpress.

management@devexpress.com

WIN WPF ASP MVC HTML5



To learn more, please visit our website →

www.devexpress.com

```
appcenter test run espresso --app "mattgibbs/Hello-App-Center" --devices
"mattgibbs/latest-android" --app-path app\build\outputs\apk\debug\app-
debug.apk --test-series "main-activity" --locale "en-US" --build-dir app\
build\outputs\apk\androidTest\debug
```

The CLI performs the upload, then lists the devices to be used for the test run. As the tests are deployed to devices, it polls the API for the test run to show progress.

The SDK provides a method for simulating a crash, making it easy to test the integration and see how the crash reporting works.

Distribute

App Center helps you distribute your app to testers and beta users before submitting it to an app store for public release. To install on iOS devices, you must first register the device with your Apple developer account and add it to a provisioning profile. When the app is signed based on that provisioning profile, it can be installed on those devices. Attempting to install it on another device will fail. App Center streamlines the process of gathering the necessary device identifiers from testers, performing the registration and signing. Android apps do not require the extra device registration step prior to signing.

Apps are shared with a distribution group. A Collaborators distribution group is automatically populated with all the users that have been added to the app as a Manager, Developer or Viewer. Distribution groups can also include testers invited by e-mail address. When you add a new tester, they're sent an e-mail inviting them to test the app. **Figure 3** shows the invitation that testers receive. By default, distribution groups require testers to log in to the App Center install site to view the list of apps that have been shared with them and see the download links. When you create a distribution group, you can choose to allow public access; the testers will not be required to log in.

Signed builds produced by App Center can be distributed directly to a distribution group. Select the build and click Distribute. You can select an existing distribution group. The release notes are pre-populated with the commit text from the build. You can update the release notes using markdown. Members of the distribution group are sent an e-mail announcing the release. The install site displays the release notes.

Crash Reporting

You can enable your app to send crash errors to App Center. The crashes are processed to identify groups of the same type of crash. The stacktrace helps you identify where in the source code the crash is occurring. The App Center SDK is modular, so that you can avoid adding overhead to your app for SDK functionality you aren't using. The SDKs have been open sourced on github.com in the Microsoft organization. To integrate App Center into your app, update the dependencies section of the build.gradle file for your app to include the following:

```
def appCenterSdkVersion = '1.0.0'
compile "com.microsoft.appcenter:appcenter-crashes:${appCenterSdkVersion}"
```

Update the onCreate method of your MainActivity to call the start method of AppCenter. One of the arguments is the application key provided by App Center. You can see it in the Settings page for the app. Additionally, you pass references to the modules you want to use. In this example, I'm enabling Crash reporting and App Center Analytics:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    AppCenter.start(getApplication(), appSecret:"your-app-key", Analytics.
class, Crash.class);
}
```

Crashes aren't sent immediately when the crash happens. The data is collected and sent when the user next launches the app. This allows you to customize the SDK behavior if you want to present the user with more information about crash reporting and get user consent to send additional data. You don't need to build special code to crash your app deliberately. The SDK provides a method for simulating a crash, making it easy to test the integration and see how the crash reporting works.

```
Crashes.generateTestCrash();
```

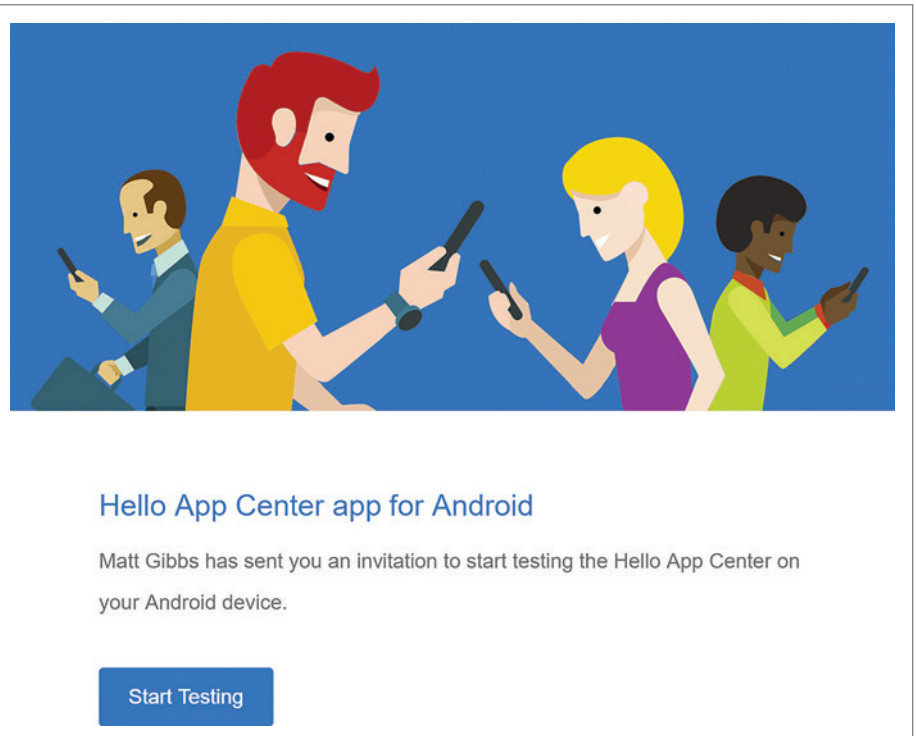


Figure 3 Tester Invitation



Rider

New .NET IDE

**Cross-platform.
Killer code analysis.
Great for refactoring.**

From the makers of ReSharper,
IntelliJ IDEA, and WebStorm.

Learn more
and download
jetbrains.com/rider



JET
BRAINS

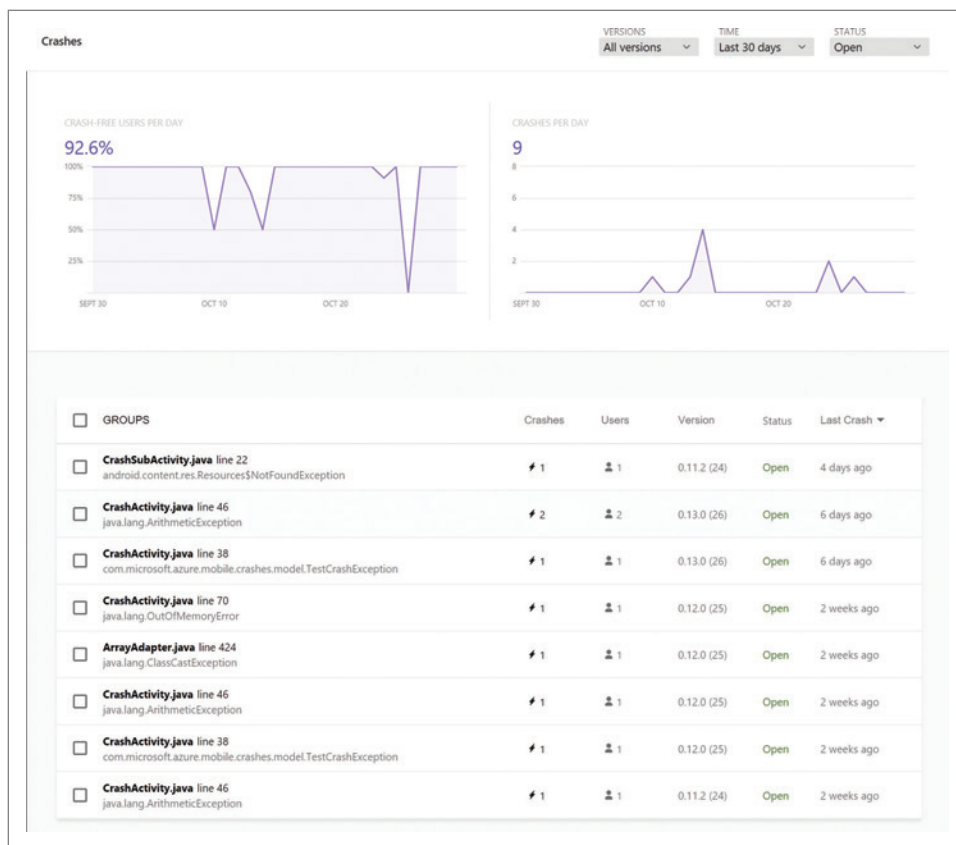


Figure 4 Crash Groups

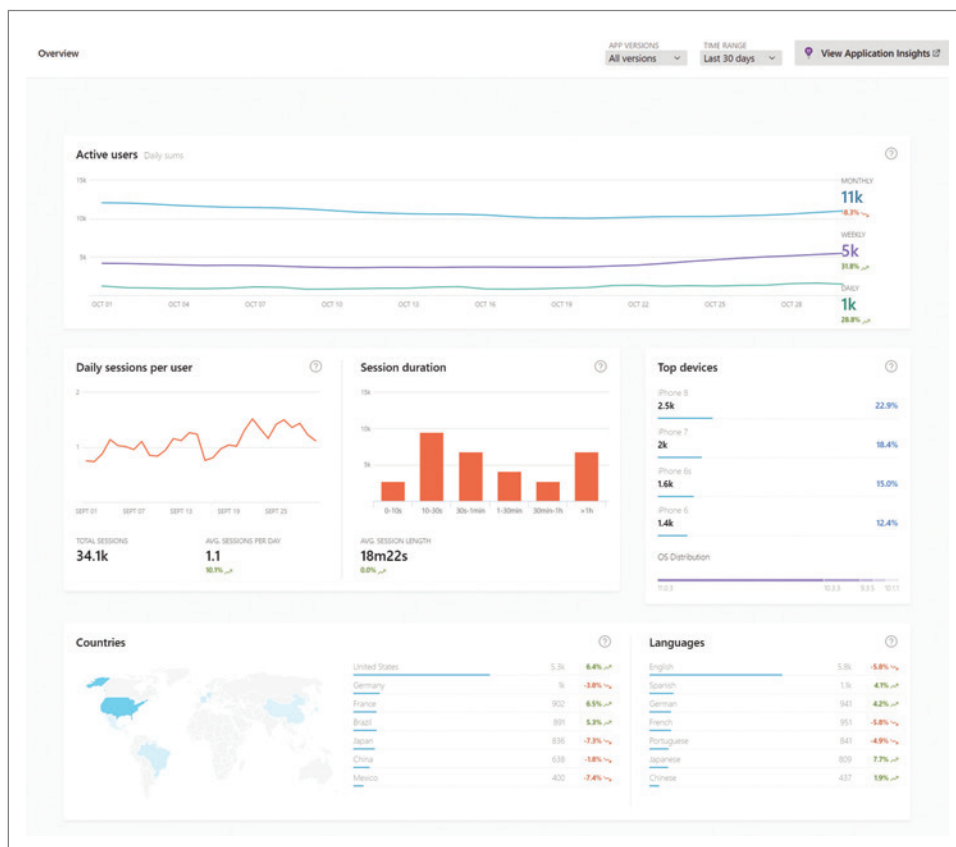


Figure 5 App Center Analytics

Figure 4 shows the Crashes area of App Center. The first chart shows how many users are experiencing crashes. The second chart shows the number of crashes received by App Center. Below the charts is a list of crash groups with the number of crashes, number of affected users, version and status. Each crash group is created with a status of open.

In the crash group details page, you can change the status to closed or ignored as you work to fix the top issues affecting users. The details page shows the stacktrace of the crash, along with distribution details of the most affected device and OS version. Individual crash reports are listed.

Click on a specific crash report to see more details. The app launch time, country, carrier, and system language setting are shown for each crash instance. If you've enabled the Analytics module, the details page will show all the telemetry events that occurred during the user session leading up to the crash.

You can have App Center send e-mail notifications whenever a new crash group is created. In the app settings area, you also have the option to provide a webhook URL that will be invoked for new crash groups or when new app versions are released. In the same way App Center can be connected to your repo service for build, it can also be connected to your issue tracker. When new crash groups are created, App Center will automatically create an issue for you. VSTS and GitHub are currently supported. Additional integrations are coming soon.

Analytics

App Center analytics helps you understand your users. The single-line integration provides information about how many users are using the app. When the app is installed, it's assigned a unique identifier used for analytics. (Don't

be surprised during development if you uninstall and reinstall on the same device and see the count of active users increase. App updates continue using the same identifier.) **Figure 5** shows the Analytics page for the fictitious Contoso Air app.

The App Center SDK is modular. To enable gathering and sending analytics data, add the module in the dependencies.

```
def appCenterSdkVersion = '1.0.0'
compile "com.microsoft.appcenter:appcenter-analytics:${appCenterSdkVersion}"
```

Override the onCreate method and start the App Center module you're using:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    AppCenter.start(getApplication(), appSecret:"your-app-key", Analytics.class, Crash.class);
}
```

App Center shows the distribution of sessions, devices, countries, languages and app version. It also provides the distribution of OS version. For Android, the API level is shown. Countries are identified by the carrier country provided by the platform. Language is based on the user's system settings. You can filter by version and time range. App Center shows distributions for 7, 30, 60 and 90 days, along with trends against the previous time period. The API allows you to query for custom date ranges up to 90 days.

App Center is mission control for your apps.

A session is started when the user first launches the app. When an app is resumed, if it was in the background for less than 20 seconds, it isn't considered a new session. This avoids incorrectly treating every short interruption as the end of a session. The session duration is calculated by looking at all App Center telemetry coming from the app. They're grouped into duration "buckets" to give you a sense of how long users are interacting with the app. If you don't add any custom events for tracking, the sessions will all be in the 0-10 second bucket unless the app crashes.

Track Events. You can instrument your app with custom events to understand exactly how the app is being used. App Center will count up to 200 unique event names per day. Event names are limited to 256 characters in length:

```
Analytics.trackEvent("Login");
```

The event analytics provide details about the count of the event, the number of users with the event, the count per user and the count per session. It also provides the changes in counts over time. You can also associate name/value properties with an event. App Center analytics will provide the distribution of the top-10 values for each property name:

```
Map<String, String> properties = new HashMap();
properties.put("category", "image");
properties.put("filetype", "png");
Analytics.trackEvent("File upload", properties);
```

Export Data. App Center analytics will keep 90 days of data. If you want to do analytics over longer periods of time or do ad hoc and custom query processing, you can configure continuous

App Center Release Automation

App Center can automate store releases. After an Android app has been published to the Google Play store for the first time, subsequent releases can be initiated from App Center. Publishing to Google Play requires a developer account. Register at bit.ly/2hx6pp4. Use the Google Play Console to first publish the app and to create an API project. You must have an API project for calling the publishing APIs.

App Center uses server-to-server authentication to publish to the store. On the API Access page of the Google Play Console, create a service account. Select Project Owner as the role with a new private JSON key type. The JSON file will be downloaded by the browser. Keep this file secure.

After the service account is created and the JSON private key generated, click Grant Access. The JSON key can now be used to authenticate as the service account with the project owner role.

In App Center, the Stores feature is in the Distribute area. When you create a connection to the Google Play store, App Center will prompt you to upload the JSON key that was generated for the service account and connect to the store. The connection will show alpha, beta and production tracks for publishing.

The next time you're ready to publish through Google Play, navigate to the Stores area of App Center, and select the destination track. App Center will soon support the option to select a package built in App Center instead of uploading it. Production track releases require the application package version be greater than the currently published version. Also note that store releases must be release builds, and signed with a key store, as well as zipaligned.

The release initially shows a status of submitted. App Center delivers the package to Google Play and the status is updated to "published." It can take up to 24 hours before the package is released and available for download.

export of the analytics data from App Center. The two options for export are Azure App Insights and Azure Blob Storage. When you setup an export to Azure App Insights, a schema for App Center data is configured. If you were previously using HockeyApp with App Insights, your existing queries can be reused. Exporting to Azure blob storage provides additional flexibility to import your data to an on-premises data warehouse or into other Big Data processing systems in Azure.

App Center is mission control for your apps. It brings together the key services you need to build better apps faster. You can have continuous integration for your app that includes running tests on real devices. You can distribute to beta users and leverage crash handling to identify and fix critical problems before releasing to the public. And you can instrument your app to get a deep understanding of how the app is used to better engage with the users. ■

MATT GIBBS works as a group engineering manager at Microsoft, delivering services and SDKs for mobile app developers. In previous years, he worked on Azure SDKs, Xbox Live Services, ASP.NET, Windows Forms, IIS and "classic" Active Server Pages.

THANKS to the following Microsoft technical experts for reviewing this article: Scott Densmore and Thomas Dohmke

WE ARE CHANGING THE WAY YOU LOGIC REPORTING



WINDOWS FORMS



WPF



ASP.NET MVC



ASP.NET AJAX



CLOUD WEB API

www.textcontrol.com

ING OOK AT

TEXT CONTROL



The Road to Continuous Delivery with Visual Studio Team Services

Willy-Peter Schaub

In today's fast-paced, feature- and value-driven markets, it's imperative that developers deliver value quickly and continuously, and react to feedback faster than the competition. Key to that effort is the practice of continuous delivery (CD), which uses continuous integration (CI) practices and automates build, test, configure and deploy activities in the DevOps cycle. CD demands that dev shops update both tooling and practices, so it's not surprising to see many organizations asking common questions, such as:

- What is CD and how can it help deliver changes and value faster?
- What tools can you use to provision CD?
- How do you integrate various tools and cloud services products in a CD effort?
- How do you mitigate the risk in a CD/CI pipeline?

This article discusses:

- Continuous delivery and how can it help you deliver changes and value faster
- How to use VSTS as the Azure DevOps solution and integrate with various other products
- Lessons learned at Microsoft through practice of DevOps and guidelines for continuous delivery

Technologies discussed:

Visual Studio Team Services (VSTS), Microsoft Azure Portal

This article aims to answer these questions, sharing insight into the lessons learned so far here at Microsoft as we practice DevOps, and explore our guidelines around CD.

It's worth noting that there's a subtle difference between CD and continuous deployment. Continuous deployment is suitable for teams with a high level of confidence in quality gates and are deploying known-good releases directly to a production environment. With CD, release moves through several environments, which may include functional, performance and staging tests and approvals along the way, before going to production. This article focuses on CD.

Continuous Delivery

CD is a lean practice that aims to eliminate diverging code, duplication of effort and, most important, merge conflicts. It starts with CI—the process of automating the build and testing of code every time anyone commits a change to version control. When a developer implements a new feature or bug fix in a feature branch, she typically submits a pull request (bit.ly/2ha4c2G) when the feature is complete. Upon approval of the pull request, changes are committed and merged into the master branch.

CD is the process to build, test, configure, deploy and confirm a feature change to production. To deliver value to users, it's important to achieve the shortest path to deploy a new feature (known as time to mitigate, or TTM) and to minimize reaction time to feedback (known as time to remediate, or TTR). The goal is to

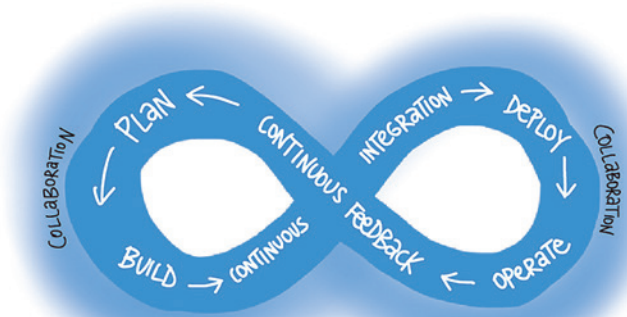


Figure 1 The Loop of Continuous Delivery

achieve an automated and reliable deployment process that consistently confirms through integration, load and user acceptance testing, and continuously releases value to your end users. **Figure 1** depicts the cycle.

CD promises to banish manual processes, time-consuming and error-prone handoffs, and unreliable releases that produce delays, cost overruns and avoidable end-user dissatisfaction.

Provisioning the Release Pipeline

A release pipeline is a collection of environments and workflows to support the activities of CI and CD. There are many products that can help you create a pipeline that supports rapid release cycles. I'll start by shedding some light on a few tools from the Microsoft stack that create and execute a CI/CD pipeline on Visual Studio Team Services (VSTS).

VSTS is a Microsoft Azure Software-as-a-Service (SaaS) that enables teams to plan better, code together and ship faster. Program in any language, including the Microsoft .NET Framework, Java, Python, Ruby and Node.js. Develop on any OS, such as Linux, macOS and Windows. And deploy to any platform, such as Azure, Amazon Web Services (AWS), Linux, Mac, Docker, Android, iOS and Windows Phone.

The Visual Studio IDE can simplify the configuration of CI and CD using the Continuous Developer Tools (bit.ly/2ha7MtE) built into Visual Studio 2017 Update 3. **Figure 2** shows how you can configure, provision and deploy your solution without leaving Visual Studio.

Azure Portal allows you to seamlessly manage the Azure environment and set up your release pipeline, as depicted in **Figure 3**.

VSTS enables you to ship more quickly, more frequently and with more confidence, while extending and customizing your release pipelines, and controlling your deployments with automatic and manual gates for approval workflows. I'll explore a release pipeline

created using VSTS in the "practical continuous delivery showcase" section. You can learn more about the CD capabilities in VSTS at the VSTS Release Management Web site (bit.ly/2zcSrNE). Also see how Bing uses CD (bit.ly/2A4m8Q5) and explore the experiences of the ALM | DevOps Rangers as they implement release pipelines for their community projects at bit.ly/2zckUWc.

Generator-team, written by Donovan Brown (bit.ly/2zclmDS) is a powerful open source Yeoman generator that demonstrates the use of the VSTS REST APIs. It creates a complete CI/CD pipeline in Team Foundation Server (TFS) or VSTS for Java, Node.js or ASP.NET, and allows you to deploy to Azure App Service, Docker to private host, Docker images in Azure App Service on Linux, and Azure Container Instances.

The Team PowerShell module, written by Donovan Brown (bit.ly/2gZfzHg) exposes portions of the REST API for VSTS and TFS. (You can learn more about the REST API at bit.ly/2h1CWjf.) It's an interesting option from the Ops perspective. It's written in pure PowerShell and can be used on macOS, Linux or Windows to connect to TFS or VSTS.

CD is the process to build, test, configure, deploy and confirm a feature change to production.

By the way, a preview of a new Common Language Infrastructure (CLI) for working with VSTS on Windows, Linux or macOS was recently shipped. It's currently geared toward developer scenarios like creating pull requests and updating work items, but will support a richer set of capabilities over time. The new CLI will also make it easier to automate interactions with VSTS using scripts. To learn more, visit Get Started with VSTS and TFS at docs.microsoft.com/en-gb/vsts.

A Practical CD and Tooling Showcase

Let's review a typical release pipeline created using VSTS, as illustrated in **Figure 4**. If you prefer to create your own instance of

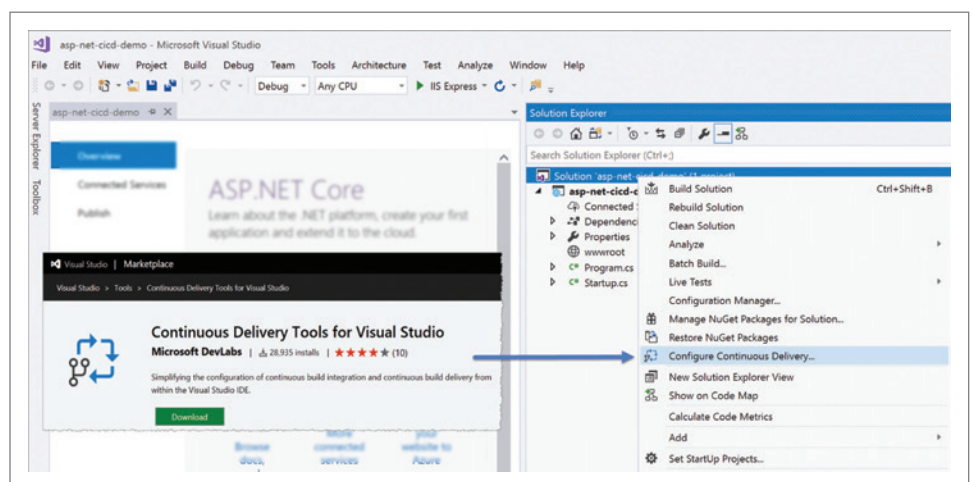


Figure 2 Continuous Delivery Tools for Visual Studio

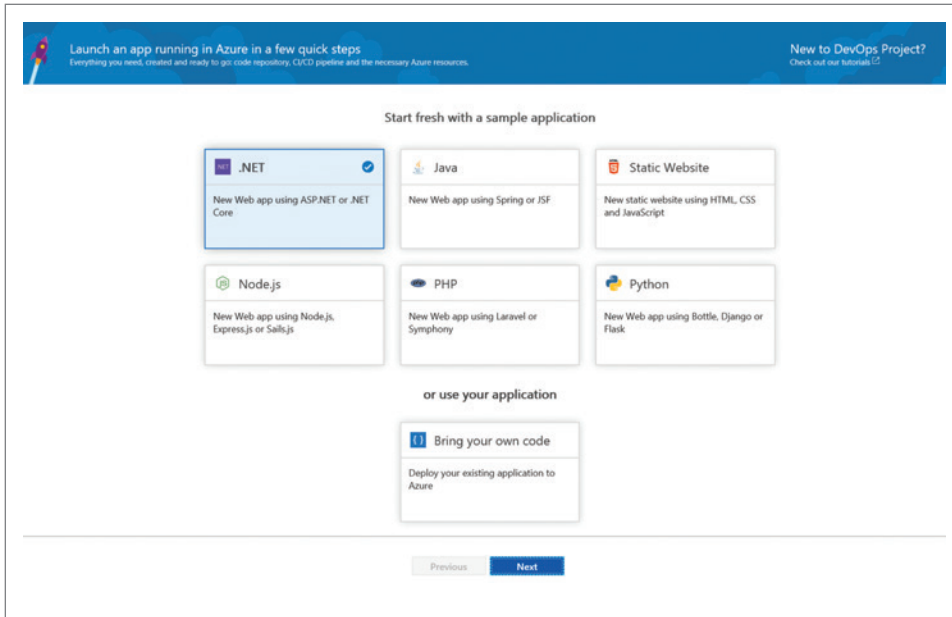


Figure 3 Deploy with Confidence from Azure

such a pipeline, read the in-depth paper, “CICD Pipeline for Your Extensions with Visual Studio Team Services,” before continuing. You can download the PDF from bit.ly/2hCAZKX.

Our journey starts with the team committing its code to a Git repository or service provider such as VSTS (Git and Team Foundation Version Control), GitHub, BitBucket or Subversion. When a developer commits changes to the team’s repository (1), continuous integration is optionally triggered (but strongly recommended) to check the associated pull request before you commit changes and after you successfully merge changes to a specified branch. VSTS not only automates your build, testing, and other validation tasks, it gives you an insight and traceability into everything in the build, including code changes, review comments, test results, and code coverage.

CD, started by the build, is the process of validating, configuring and deploying your changes to one or more environments. You can choose to have a predefined schedule and conditions to trigger the deployment for each environment in your release pipeline. In addition, you can define approvers for each environment before and after deploying an environment. Approvals can be set to be automatic or can be executed by one of many specific users, all users in any order, or all users in sequential order. When a group is specified as an approver, only one of the users in that group needs to act.

In the example for this article, we’re automatically processing the Canaries environment (2), which privately deploys the extension to the Visual Studio Marketplace (3). We’re using private extensions initially, so that they’re not discoverable on the public marketplace. Private extensions are in production for the customer, but only visible to the customer’s nominated VSTS accounts. See the previously mentioned white paper for details on the publishing process. It’s important to emphasize that your release pipeline could deploy to a variety of destinations, but we’re targeting the Marketplace as we are deploying a VSTS extension in this example.

CD proceeds with a parallel (forked) deployment to the On-Prem Validation (4) and the Early Adopters (5) environments. The former proceeds automatically, as indicated by the pre-deployment approvers icon (⌘) in the pre-deployment condition. The latter introduces the pre-deployment manual approvers icon (⌘), which shows that third-party approval is needed before the deployment can begin. In addition, the example specifies that the environment trigger fires only if changes apply to the master branch (6), effectively shielding production from other branches.

You can implement several control goals to progressively expose your changes to subsequent environments, keeping an existing

(blue) version live, while deploying and validating a new (green) version. If an issue is discovered, the pipeline allows you to stop the deployment to minimize impact to end users.

VSTS not only automates your build, testing and other validation tasks, it gives you insight and traceability into everything in the build, including code changes, review comments, test results and code coverage.

CD continues with a joined deployment pipeline, where deployment to the Users environment (7) occurs only after successful deployments to both the Early Adopters and On-Prem Validation environments. See “Phases in Build and Release Management” (bit.ly/2zLrL71) for details on running tasks on different agents, manual interventions, and conditions under which tasks will process or stop your release pipeline.

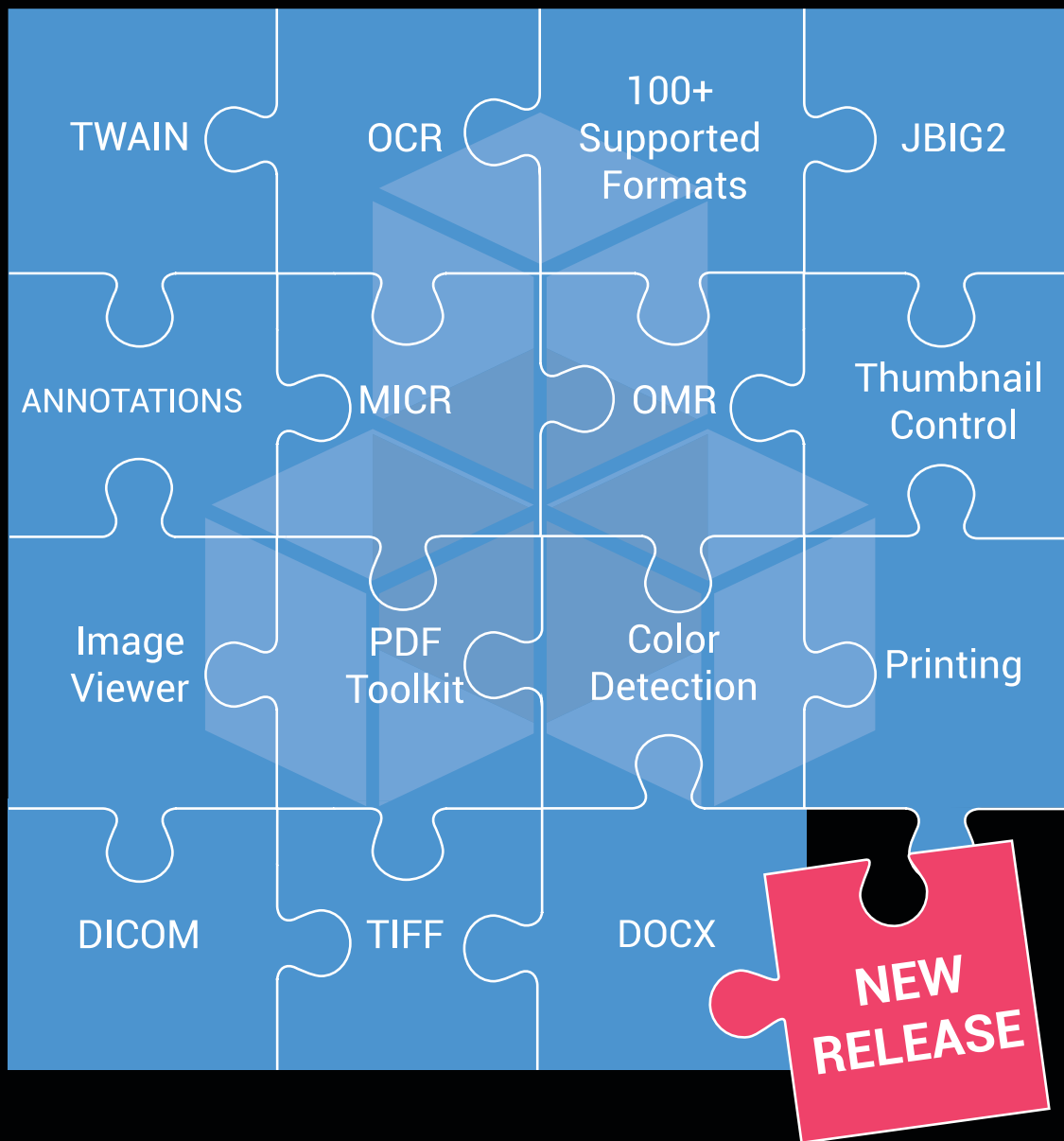
We’ve used deployment rings to limit impact on end users, while gradually deploying and confirming change in the production environments. We evaluate the impact, sometimes called the “blast radius,” through observation, testing, diagnosis of telemetry and, most important, user feedback. The Canaries environment is the first stage in the ring deployment model we’re using to expose the private extension to a controlled user group to test the new version

GdPicture.NET



100% ROYALTY FREE

Imaging SDK For WinForms, WPF And Web Development



Leverage your apps. with **GdPicture.NET** Imaging Toolkit

DOWNLOAD
YOUR FREE TRIAL

www.gdpicture.com

GdPicture.NET is an



product

in production before a broader rollout. Again, the article “Phases in Build and Release Management” offers insight (bit.ly/2zLrL71).

As discussed, you can orchestrate deployments across multiple environments, using manual or automated approval gates, while supporting high availability for your application. Whether you’re targeting the cloud, the hybrid cloud, or an on-premises environment, VSTS is the recommended Azure DevOps solution. It allows you to plan, monitor, build and deploy to Windows, Unix, Android or iOS, from .NET, Java, PHP, Python, Ruby, C++, as well as from many other products such as Ant, Maven, Gradle, msbuild, Junit, NUnit, xUnit, MSTest and Jasmine. **Figure 5** shows the range of available solutions.

Both TFS and VSTS support a rich integration model using the same marketplace extension. If you cannot find a task needed for your release pipeline, you can build custom tasks that integrate with your CI and CD. For example, we’re using the open source VSTS Developer Tools Build Tasks extension (bit.ly/2hcqopf) to package and publish our extensions to the Visual Studio Marketplace. Another example, the AWS Tools for Microsoft VSTS extension (bit.ly/2ybJYZZ), delivers tasks for Amazon S3, Beanstalk, CodeDeploy, Lambda and more. It enables you to orchestrate deployments that span both the Azure and AWS clouds.

Mitigating Risk in a CI/CD Pipeline

So far, I’ve explored how VSTS is a pipeline system with a simple, real-world example. The risk of failure grows rapidly as you increase the number and frequency of releases, and add dependencies and complexity to pipelines. Let’s look at a few techniques for gaining visibility into real-time status and mitigating the risk in your CI/CD pipelines.

Garbage in garbage out. The old saying applies to your release pipeline! A reliable and quality release pipeline depends on a healthy source to feed its CI and CD. You can leverage proven branching strategies, like those described in “Adopt a Git Branching Strategy” (bit.ly/2A0QQEA) and in “Branching Strategies with TFVC” (bit.ly/2yaqEMo). You can also employ branch policies, and pull requests to maintain a healthy and shippable master branch.

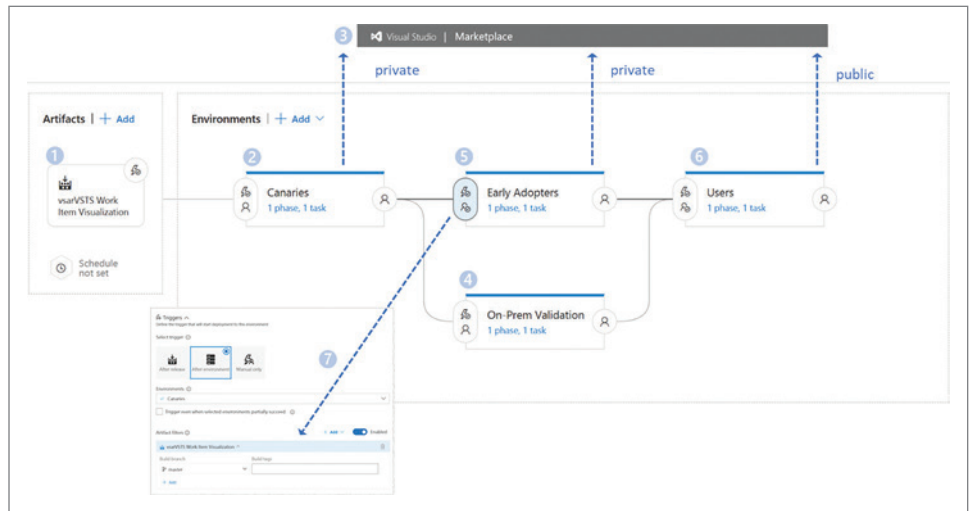


Figure 4 View of a Typical Continuous Integration/Continuous Delivery Release Pipeline

Figure 6 summarizes a few lessons from our DevOps transformation, documented by Sam Guckenheimer at bit.ly/2j7x0JY. For example, we work in three-week sprints (1), using this cadence as our common heartbeat to plan and deploy continuously. We manage debt continuously (2), to avoid the accumulation (or spike) of debt and resultant merge complexity (3) that we saw in the past.

The master branch (4) is our single source of truth, always in a healthy and shippable state. Teams use short-lived feature branches to isolate their work and submit a pull request when the feature is complete (5). On approval of the pull request, we merge the feature into the master branch. Teams repeat the process continuously for added work. The release isolation strategy (6) introduces one or more release branches from the main, enabling concurrent release management, multiple and parallel releases, and exact snapshots of our existing (blue) version at release time.

Pre-merge validation is important! Establish branch policies to ensure that the master branch meets your desired quality criteria. Always encourage developers to submit code changes

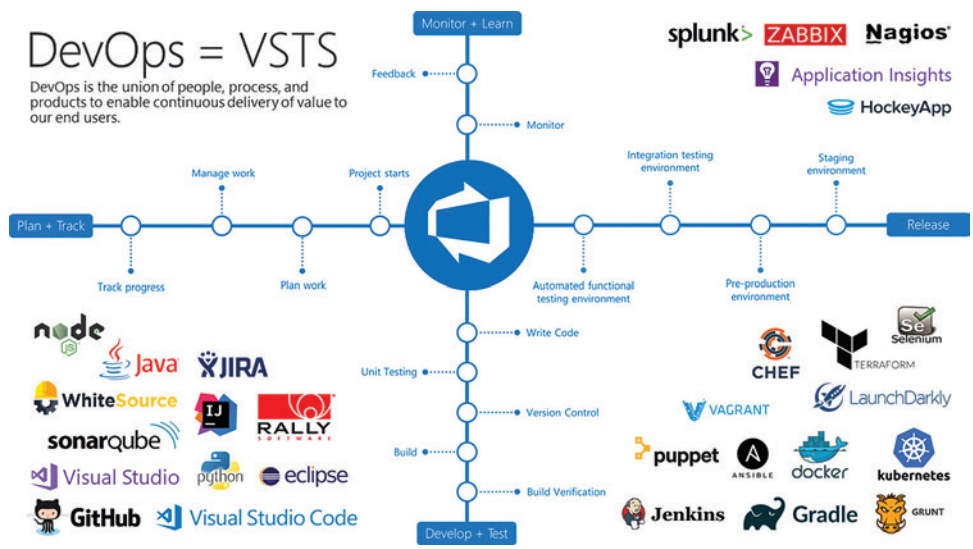


Figure 5 Visual Studio Team Services Is Our Backbone Azure for DevOps Solution



Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn
magazine

msdn.microsoft.com/flashnewsletter

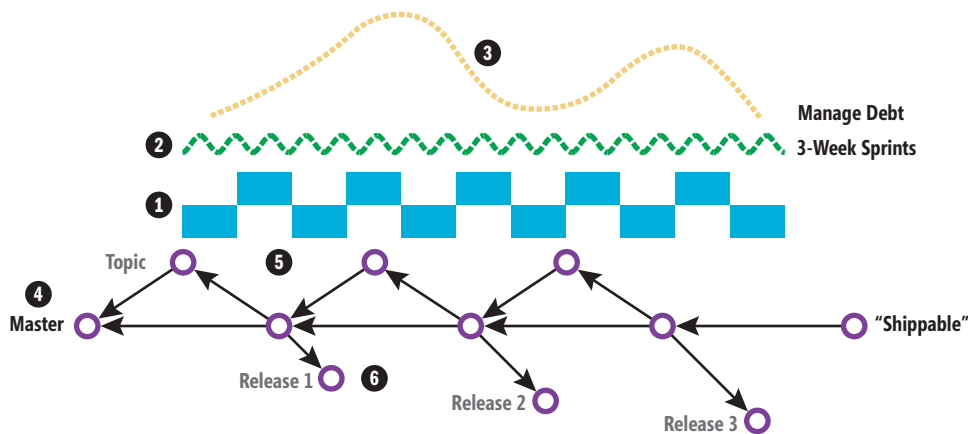


Figure 6 Keep Your Master in a Good and Shippable State

via pull requests. You can selectively enforce policies, for instance that pull requests must have approval from a specified number of reviewers, or that work items are linked to improve traceability. Resolution of comments and a specific merge strategy upon completion are two more examples. Beyond the pre-merge validations, you can add build policies to automatically trigger a build that must succeed to complete pull requests, and invalidate builds when the master branch changes.

Security is paramount. It's vital that security be confirmed continuously throughout the process, that applications are safe, and that data stays secure and private. Here are just a few of the ways that DevOps and an automated pipeline can work with CD to improve security:

- Simulate attacks and system stress during the build to detect vulnerabilities and failures. Take remedial action or block releases as necessary.
- Create a dynamic infrastructure to avoid a persistent place for vulnerabilities to hide. Quickly recreate the infrastructure and re-deploy new packages when vulnerabilities are detected.
- Automate tests for security and continuously monitor in production to ensure your solution is secure.

Check out the article “Adding Continuous Security Validation to Your CI/CD Pipeline” for an in-depth exploration of how security should be implemented in a CI/CD scenario at bit.ly/2j6GHls.

Vet your open source components. Tools such as WhiteSource (whitesourcesoftware.com) can be immensely useful. It integrates seamlessly into the release pipeline and continuously checks the security, licensing and quality of open source components. The ALM | DevOps Rangers have been dogfooding this tool with all its VSTS extension release pipelines and shared its outcomes at bit.ly/2yD8yqU.

Use feature flags. Feature flags help keep feature branches shorter and let you integrate them into the master more often. Feature flags and the ring deployment model are symbiotic. Rings limit the exposure (blast radius) to predefined user groups, while feature flags provide a granular way to expose or hide a specific feature to all or selected users. Feature flags are invaluable for delivering value and receiving feedback on features easily, quickly and continuously.

You roll out code across the entire ring topology before you enable the feature flag. This allows you to release and test features while still in development, or to switch on verbose telemetry in produc-

tion with a flip of a flag. See “Phase the Features of Your Application with Feature Flags” (bit.ly/2yD8yqU) for a discussion of feature flags and A/B testing with our release pipelines, using the LaunchDarkly (launchdarkly.com) Software as a Service.

Pipeline health visualization.

Visualize the health of your CI builds using the Team Project Health extension (bit.ly/2hgFk9A), which adds a visual cue to your dashboards informing you about the status of your builds and releases. Codify/build-light (github.com/Codify/build-light) is a Raspberry Pi-based solution

that visualizes the health of your CI builds.

Greenlighting feature. New to VSTS is greenlighting, which intelligently automates approvals. It lets you define a set of gates in your pre- and post-deployment options that integrate signals from monitoring systems and other external services. Greenlighting not only visualizes your pipeline health, it uses gates to determine RED/GREEN/BLUE status based on real-time signal sampling.

New to VSTS is
greenlighting, which intelligently
automates approvals.

Greenlighting can trigger an Azure function to ensure a successful completion, match work item query results to thresholds, and define a time and schedule, among a host of other things. By greenlighting your pipeline, you enable automation and faster feedback loops. You're able to focus on your deployment speed, supporting faster time to mitigate issues and faster time to value.

Wrapping Up

CD is an ongoing and endless process of innovation. Now that we've covered the concepts of CD, you should feel confident exploring ways to improve your release pipelines. It's all about taking incremental steps to improve your process. Reducing time to test and time to deployment, minimizing integration debt, and raising release quality may feel like a daunting challenge. By taking small, early steps, it all becomes achievable. DevOps isn't a destination, it's a journey of continuous, rapid improvement. ■

WILLY-PETER SCHAUB is a program manager in VSTS, working at Microsoft Vancouver in beautiful British Columbia. Since the mid-'80s, he's been striving for simplicity and maintainability in software engineering. You can follow him on LinkedIn aka.ms/willysli or Twitter at [@wpschaub](https://twitter.com/wpschaub).

THANKS to the following Microsoft technical experts for reviewing this article: Ryan Britton, Rui Carrilho de Melo, Edward Fry, Vijay Machiraju, Tiago Pascoal and Martin Woodward

Manipulating Documents?

APIs to view, convert, annotate, compare, sign, assemble and search documents in your applications.

Try GroupDocs APIs for FREE



GroupDocs.Viewer

View over 50 documents and image formats in any application using document viewer APIs.



GroupDocs.Annotation

Add annotations to specific words, phrases and any region of the document.



GroupDocs.Conversion

Fast batch document conversion APIs for any .NET, Java or Cloud app.



GroupDocs.Comparison

Compare two documents and get a difference summary report.



GroupDocs.Signature

Digitally sign Microsoft Word, Excel, PowerPoint and PDF documents.



GroupDocs.Assembly

Document automation APIs to create reports from templates and various data sources.

▶ GroupDocs.Metadata

▶ GroupDocs.Search

▶ GroupDocs.Text

▶ GroupDocs.Editor



Americas: +1 903 306 1676
EMEA: +44 141 628 8900
Oceania: +61 2 8006 6987
sales@asposeptyltd.com



Download a Free Trial at
<https://downloads.groupdocs.com/>

Create Serverless APIs Using Azure Functions

Alex Karcher

In the world of API development, serverless application platforms have redefined the problems developers must solve. With instant scale and consumption billing, developers are increasingly turning to serverless tools like Azure Functions to reduce development time, be more adaptive to changing traffic patterns, and to stop paying for overprovisioned infrastructure.

Serverless simply means that the developer doesn't have to consider the underlying application server when writing code. You write a single class, with a run method, and then define triggers that execute that code. As a trigger fires multiple times in close succession, your code is loaded onto more and more workers to handle the load. The scale out happens on the sub-second level, and you're only charged for the CPU running time plus the allocated memory of your function. There's no charge for idle Functions.

Think of serverless like wireless. There are still wires behind your wireless router, but the consumers of your wireless service don't

have to manage or configure them. With serverless, you don't have to manage or configure servers to run your code.

In Azure Functions, the abstraction of the application server is accomplished through a powerful trigger and binding framework. Triggers and bindings are a declarative way to define how a function is invoked and with which data it works. A trigger defines how a function is invoked. A function must have exactly one trigger. Triggers have associated data, which is usually the payload that triggered the function.

Input and output bindings provide a declarative way to connect to data from within your code. Similar to triggers, you specify connection strings and other properties in your function configuration. Bindings are optional, and a function can have multiple input and output bindings.

Using triggers and bindings, you can write code that's more generic and doesn't hardcode the details of the services with which it interacts. Data coming from services simply become input values for your function code. To output data to another service (such as creating a new row in Azure Table Storage), use the return value of the method. See **Figure 1** for a listing of all supported triggers and bindings.

Azure Functions is particularly well suited to applications with bursty workloads, like orchestration and automation tasks. In the Infrastructure-as-a-Service (IaaS) world, these infrequently requested tasks are often bundled in with other services to maximize virtual machine (VM) usage, or run on expensive dedicated VMs that are idle most of the time. For example, a synchronization service might get 10,000 requests once a day, exactly at midnight; Azure Functions is perfectly suited to handle that uneven load. The

This article discusses:

- Why you should choose serverless
- General availability of Azure Functions Proxies, and the rest of the Azure Functions toolbox
- Best practices, design tips and step-by-step guide to building a serverless API
- Advanced tools and techniques

Technologies discussed:

Azure Functions, Azure Functions Proxies, REST APIs

Figure 1 Azure Functions Triggers and Bindings

Type	Service	Trigger*	Input	Output
Schedule	Azure Functions	✓		
HTTP (REST or Webhook)	Azure Functions	✓		✓**
Blob Storage	Azure Storage	✓	✓	✓
Events	Azure Event Hubs	✓		✓
Queues	Azure Storage	✓		✓
Queues and Topics	Azure Service Bus	✓		✓
Storage Tables	Azure Storage		✓	✓
SQL Tables	Azure Mobile Apps		✓	✓
NoSQL DB	Azure Cosmos DB	✓	✓	✓
Push Notifications	Azure Notification Hubs			✓
Twilio SMS Text	Twilio			✓
SendGrid E-Mail	SendGrid			✓
Excel Tables	Microsoft Graph		✓	✓
OneDrive Files	Microsoft Graph		✓	✓
Outlook E-Mail	Microsoft Graph			✓
Microsoft Graph Events	Microsoft Graph	✓	✓	✓
Auth Tokens	Microsoft Graph		✓	

(* The HTTP output binding requires an HTTP trigger.)

function will parallelize out to meet that incoming demand, and further reduce the wall clock runtime of the operation.

Azure Functions Proxies

One of the most popular uses for Azure Functions is API hosting. With that in mind, we created Azure Functions Proxies to allow developers to separate their API functionality across multiple function apps, without having to spread their API across multiple domains. Azure API management provides the same ability to composite together APIs; however, it's much pricier because of the extra functionality that would go unused in this scenario. This past spring we released Azure Functions Proxies in preview, and this month we're releasing Azure Functions Proxies to general availability. It has reached release quality, and is now ready to host full production workloads.

Azure Functions Proxies provides a core set of API development tools specifically suited for the serverless API developer. First, Azure Functions Proxies allows you to composite multiple APIs across functions and services together into one, unified API surface. Second, Azure Functions Proxies enables hosting mock API endpoints to quickly get started developing against an API. Third, Azure

Functions Proxies allows any API to be routed through functions, to leverage metrics, security and OpenAPI definition support. Finally, Azure Functions Proxies allows static content to be hosted on a function domain.

Building a Serverless API

This tutorial will make ample use of public APIs as integration points, in lieu of Azure services. The Azure Functions documentation has many walk-throughs for accessing just about any resource through Azure Functions, or Logic Apps, so I'll stick to the API examples.

Mocking an API in Azure Functions Proxies It's day one of your new project, and the most important task for you, as an API developer, is to validate the API design, and get enough implemented that the mobile devs on your team can get started building a client for the API. A mock API in Azure Functions Proxies will allow you to quickly create a live API, to visualize the whole API surface and unblock the mobile team.

Azure Functions Proxies enables you to host mock APIs by adding response override rules, and no back-end URL. A proxy will respond to matching a request with whatever sample data you've given. This will allow

you to expose a live API in a matter of minutes to unblock your partner team developing against that API. As you complete functionality, you can replace individual proxy rules without requiring the partner team to update their code.

Azure Functions Proxies provides a core set of API development tools specifically suited for the serverless API developer.

To start, you'll need to navigate to the Azure portal and create a new function app with the default configuration. Once you're in your blank app, click on the plus sign to the right of Proxies and fill out the form to match **Figure 2**. This will configure a mock API proxy, which returns a simple JSON object and appropriate header when sent a GET request to the /mobile-user endpoint.

Serverless API Design Tips

Stick to one function per operation. There's no cost penalty to have many idle functions, and you'll end up with more portable and reusable code.

Use Software-as-a-Service offerings whenever possible. Any service you don't have to manage saves developer time, and provides one less regression point.

Long-running functions should delegate. You're paying for code that's async awaiting responses, so use durable functions or queues to delegate that work to another function, and call a final function on task completion.

Connect services with OpenAPI definitions. OpenAPI allows you to programmatically define your API surface and onboard new API consumers. OpenAPI powers client SDK generators for most languages, as well as a wide range of documentation generators.

HTTP communications between functions must be secured. Each HTTP endpoint is Internet-addressable, so Azure Functions natively supports service principle authentication to secure HTTP traffic between functions. Alternatively, you can use any other trigger to avoid HTTP entirely, or put your functions in an App Service Environment to secure communications.

Figure 2 Configuring a Mock API in Azure Functions Proxies

Use Postman, or your preferred API testing tool, to send a GET to the proxy URL. You should receive back your formatted JSON object in the response body, like so:

```
{
  "Name": "Proxies Connect"
}
```

You can add as many request overrides as you want, to create more complex hardcoded mock APIs with different headers, status codes or body contents.

Creating an Interactive Mock API The mobile devs have tested a basic connection with your static mock API, but now they want something with dynamic content. We'll create a new mock API that allows them to test out a POST operation. They'll POST a username

Figure 3 An interactive Mock API

```
"MobileUserPOST": {
  "matchCondition": {
    "route": "/mobile-user",
    "methods": [
      "POST"
    ]
  },
  "responseOverrides": {
    "response.statusCode": "200",
    "response.statusReason": "OK",
    "response.body": {
      "headers": {
        "User": "request.headers.User"
      }
    },
    "response.headers.Content-Type": "Application/JSON"
  }
},
```

and receive that username back.

Create a new proxy, but this time click on the Advanced editor. This will open up the Monaco editor, and allow you to directly edit the proxies.json file. Proxies.json sits at the root directory of your function app, and controls the proxy rules shown in the UI. Paste in the code from Figure 3 above the mobile-user proxy, noting proper commas and brackets in the JSON object.

Note that you can use `request.headers.headername` to insert request header values into a response. This is perfect for making more interactive mock APIs.

This proxy will create a new mock endpoint at /mobile-user that responds to POST requests. The new endpoint will return a response body with the user that the client posted as a request header. Send a POST request to the /mobile-user endpoint using Postman, and add a User header with your own username. The response

back should contain that username in the response body, like so:

```
{
  "headers": {
    "User": "MyUsername"
  }
}
```

Utilizing the request parameter, you can insert all sorts of request info into the response: `Request.method`, `request.headers.<headername>`, and `request.querystring.<paramatername>` are all available to you in the response overrides.

Moving forward implementing the API, it's important to note that request and response overrides are additive, overwriting any

Figure 4 The API Response to the Proxied /Mobile-User Endpoint

```
{
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip,deflate",
    "Cache-Control": "no-cache",
    "Connection": "close",
    "Disguised-Host": "connectfndemo.azurewebsites.net",
    "Host": "httpbin.org",
    "Max-Forwards": "9",
    "Postman-Token": "5541efc6-709a-4e95-b782-cbc946452564",
    "User": "MyUsername",
    "User-Agent": "PostmanRuntime/6.4.1",
    "Was-Default-Hostname": "connectfndemo.azurewebsites.net",
    "X-Arr-Log-Id": "60cf1436-ca3d-4408-801e-8cf47afd4a46",
    "X-Arr-Ssl": "2048|256|C=US, |CN=*.azurewebsites.net",
    "X-Ms-Ext-Routing": "1",
    "X-Original-Url": "/mobile-user",
    "X-Site-Deployment-Id": "ConnectFnDemo",
    "X-Waws-Unencoded-Url": "/mobile-user"
  }
}
```


Free AppFabric Wrapper
Quick AppFabric Migration to NCache



Extreme Performance Linear Scalability



Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing



NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

100% Native .NET

Open Source

matching object and passing along all other objects. If you want to remove an object entirely, you need to overwrite it with a blank entry.

Replace the Mock API with a Real API Now that the partner team has been unblocked, it's time to add a real API into the solution. To start off, the real API will be the httpbin.org/headers endpoint. This testing endpoint returns a dictionary of HTTP headers, so it'll be perfect to replace the `/mobile-user` POST endpoint. However, the httpbin.org/headers endpoint only accepts GET requests, so use a proxy request override to send the back-end service the correct request.

Replace the `MobileUserPOST` proxy with the following rule:

```
"MobileUserPOST": {
  "matchCondition": {
    "route": "/mobile-user",
    "methods": [
      "POST"
    ]
  },
  "backendUri": "https://httpbin.org/headers",
  "requestOverrides": {
    "backend.request.method": "get"
  }
}
```

Now resend the Post request to the `/mobile-user` endpoint. You should receive a complete list of headers, along with the name header you sent. Your response should somewhat match **Figure 4**.

Composite an Additional API Part of the strength of Azure Functions Proxies is the ability to combine multiple separate API endpoints into one outward-facing API endpoint. For the next example, we'll add a function-hosted API into the `/mobile-user` API.

Start by creating a new function app and creating a new HTTP triggered function inside of the new app. We'll utilize the built-in HTTP sample code that takes in a name as a query parameter and returns "hello name."

When you proxy the function, you're not going to want to hard-code the back-end function URL and API key into the proxy rule, so we'll use an application setting to store that value, and retrieve that value in the proxy.

In the function editor of the back-end function, copy the function URL—API key and all—to the clipboard for later. Now navigate back to the function hosting the proxies, and add an application setting with the function URL you just copied. The app settings page is under `functionname | Platform features | Application settings | + Add new setting`. Call the new setting "backendfunction" and hit save.

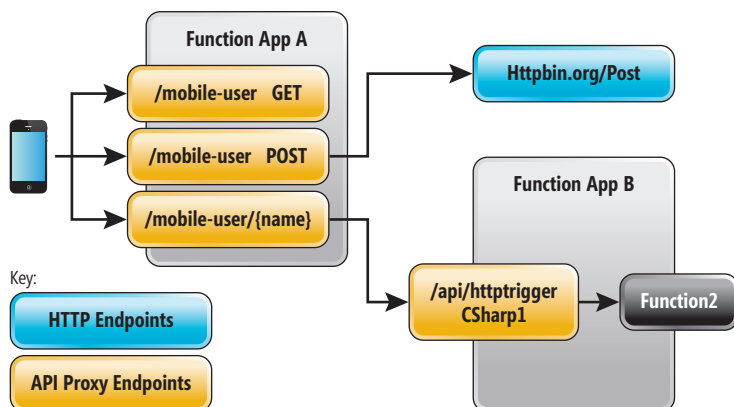


Figure 5 Final Proxy API Topology

Paste the following proxy into the advanced editor, to create a rule that proxies `/mobile-user/name` to the back-end function, to ultimately return "hello name," like so:

```
"MobileUserHello": {
  "matchCondition": {
    "route": "/mobile-user/{user}"
  },
  "backendUri": "%backendfunction%",
  "requestOverrides": {
    "backend.request.querystring.name": "{user}"
  }
}
```

Tip: Use application settings to store secrets outside of your proxies, so that the proxies.json can be checked into source control, shared widely or simply updated dynamically. Use `%appsettingname%` to reference an application setting.

When it's all said and done, your finished API topology will look like **Figure 5**.

Advanced API Topologies with APIM

For many developers, Azure Functions Proxies just scratches the surface of its API integration needs; instead, it makes sense to step up to API Management. When compositing multiple APIs together, Azure API Management (APIM) provides a much more expansive set of transformation rules, as well as policies for rate limiting and security. APIM also allows APIs to be monetized or restricted using a developer onboarding portal. In large APIs the topology is usually a single APIM endpoint resolving to multiple Azure Functions Proxies endpoints, which then have functions and Software-as-a-Service services under them.

Adding Extra Functionality

Azure Functions Proxies enables you to add a whole suite of functionality to any existing API. You can expose an OpenAPI definition, layer on authentication/authorization to provide a secured API endpoint, or measure traffic on an API using App Insights.

Host a Static Page Using Azure Functions Proxies: Azure Functions Proxies isn't just useful for API compositing; it can also be used for general HTTP redirection. You can host static HTML on your function app by either proxying the back-end URL of a service hosting HTML, or creating a proxy that returns HTTP 302 to redirect the user to a Web page. This combo can be used to host single-page applications, and is especially useful if the API endpoints used by the single-page application is in the same proxy.

Monitor a Legacy API with Application (App)

Insights: Azure Functions Proxies can be used to add lots of Platform-as-a-Service-like functionality to an existing API, such as monitoring with App Insights. App Insights is a strong monitoring suite that can calculate complex metrics on API traffic, send alerts on certain conditions, and provide a real-time analytics stream.

To enable App Insights monitoring, simply create a new function app and toggle App Insights to on at the creation blade. Then add a proxy, as simple as one route with a back-end URL. Send some requests to your proxy to have data to measure; then navigate to the App Insights resource with the same name as your function app in the portal. You'll be able to see the latency, request count, and

status codes in a live metrics stream, and query up to 90 days of historical data in the analytics portal.

For example, enter the following query into the analysis portal to see what days of the week your API receives the most requests:

```
requests
| where timestamp > ago(90d)
| summarize count() by dayofweek(timestamp)
```

Debug Proxies: Azure Functions Proxies has the ability to generate debug traces upon request. They allow you to see what rules were applied to a particular request, along with the request info. Send a request to a proxies endpoint along with the header Proxy-Trace-Enabled set to true, and you'll receive a header back with a link to the trace file. You can also set debug: true for a proxy in proxies.json to generate a debug trace for every request. The trace is stored under the function's blob storage, which is only accessible to members of your Azure subscription.

Run Azure Functions Proxies Locally: As a part of the general availability of Azure Functions Proxies, you can now run it locally, using the Azure Functions Core Tools. Find them at aka.ms/functionscoretools.

Create proxies locally with the following line:

```
func proxy create [--name] [--route] [--methods] [--backend-uri]
```

And then run those proxies by typing:

```
func host start
```

You can then send API requests to the local function endpoint using Postman, Fiddler and so on. See a full list of commands and samples at aka.ms/functionscoretools.

When proxying functions in the same function app, you can use the localhost back-end URI to reference a local function. This allows you to have one unified keyword to proxy the same function locally and when deployed to the cloud. Localhost in Azure Functions Proxies is a keyword, so you don't have to worry about appending the correct port when running locally.

Here's an example of a simple proxy to a function in the same function app:

```
"LocalFunction": {
  "matchCondition": {
    "route": "/localfn1"
  },
  "backendUri": "https://localhost/api/httptriggercsharp1"
}
```

Give Functions Proxies a Try

We're very excited for the general availability of Azure Functions Proxies, and hope you'll give it a try. Azure Functions has a generous charter of 1 million free executions and 400,000 free Gbps per month, giving you no reason to hold off on playing with Azure Functions Proxies. Serverless has been gaining traction fast in cloud native development, and now is the perfect time to jump on board.

We love feedback, and you can reach out to the product team on Twitter @azurefunctions, or submit a GitHub issue with any feature requests or issues at github.com/Azure/Azure-Functions. ■

ALEX KARCHER is a program manager on Azure Functions, working on API tools such as Proxies and OpenAPI. He's on Twitter: @alexkarcher.

THANKS to the product team for their technical expertise in reviewing this article: Matthew Henderson, Galin Iliev, Eduardo Laureano, Omkar More, Hamid Safi, Colby Tresness



Instantly Search Terabytes of Data

across a desktop, network, Internet or Intranet site with dtSearch enterprise and developer products

Over 25 search features, with **easy** **multicolor** **hit-highlighting** options

dtSearch's document filters support popular file types, emails with multilevel attachments, databases, web data

Developers:

- APIs for .NET, Java and C++
- SDKs for Windows, UWP, Linux, Mac and Android
- See dtsearch.com for articles on faceted search, advanced data classification, working with SQL, NoSQL & other DBs, MS Azure, etc.

Visit dtsearch.com for

- hundreds of reviews and case studies
- fully-functional evaluations

The Smart Choice for Text Retrieval®
since 1991

dtsearch.com 1-800-IT-FINDS

VISUAL STUDIO LIVE! (VSLive!™) is celebrating 25 years as one of the most respected, longest-standing, independent developer conferences, and we want you to be a part of it.

Join us in 2018 for #VSLive25, as we highlight how far technology has come in 25 years, while looking toward the future with our unique brand of training on .NET, the Microsoft Platform and open source technologies in seven great cities across the US.

March 12 – 16, 2018

Bally's Hotel & Casino



Respect the Past.
Code the Future.



Las Vegas

April 30 – May 4, 2018

Hyatt Regency Austin



Code Like It's 2018!



Austin

June 10 – 14, 2018

Hyatt Regency Cambridge



Developing Perspective.



Boston

SUPPORTED BY



Visual Studio
MAGAZINE

PRODUCED BY



August 13 – 17, 2018

Microsoft Headquarters



Yesterday's Knowledge;
Tomorrow's Code!



September 17 – 20, 2018

Renaissance Chicago



Look Back to
Code Forward.



NEW LOCATION!

October 8 – 11, 2018

Hilton San Diego Resort



Code Again for
the First Time!



December 2 – 7, 2018

Loews Royal Pacific Resort



Code Odyssey.



CONNECT WITH US



twitter.com/vslive –
@VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!

vslive.com

#VSLIVE25

SQL Operations Studio: Cross-Platform SQL Server Management

Julie Lerman

Have you ever been setting up a new Windows machine and dreaded having to download and install database tools that are several gigabytes in size? Or, on the other hand, wanted to do some work on your SQL Server database from a Mac and wished you had SQL Server Management Studio (SSMS)? Maybe you've been using Visual Studio Code (VS Code) and have thought, "Gee, it would be so cool if there could be a version of SSMS like this! Cross-platform, lightweight, extensible and free!" A nerd has her dreams, I guess.

Well, friends, dreams do sometimes come true. I've already had a lot of fun working with the mssql extension for VS Code, but that extension is mostly for executing SQL, though it also has some great features for visualizing query results. And it made me wish for more, such as, "If only I could browse through my database schema with an explorer like the one in SSMS or SQL Server Data Tools (SSDT) in Visual Studio." Or, "If only I didn't have to go look up the TSQL to do a database backup."

This article discusses:

- Microsoft SQL Operations Studio IDE
- Make your first server connection
- Easy access to stats and management tasks with the dashboard
- Interacting with data
- Customizing the dashboard

Technologies discussed:

SQL Operations Studio, mssql Extension for Visual Studio Code

The realization of these dreams comes via the new Microsoft SQL Operations Studio. It seems that with the mssql extension for VS Code, the SQL Server Tools team was just getting warmed up. This team is very focused on having its upcoming tools be not

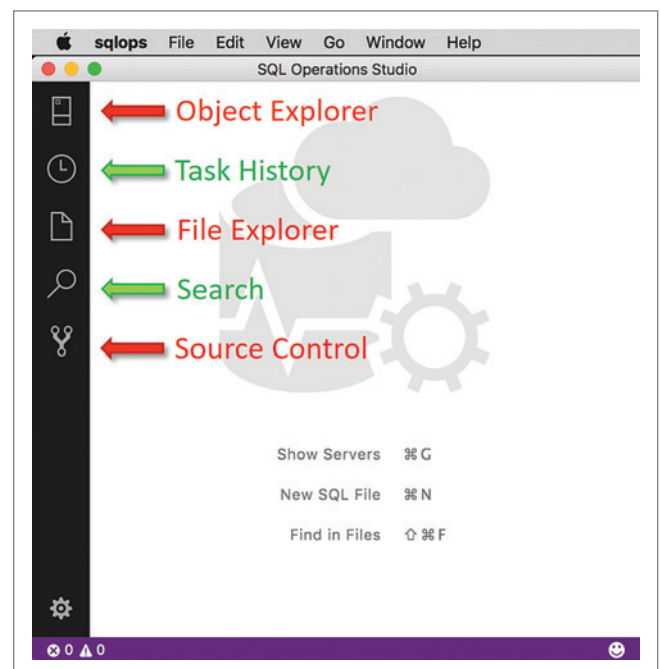


Figure 1 The SQL Operations Studio UI

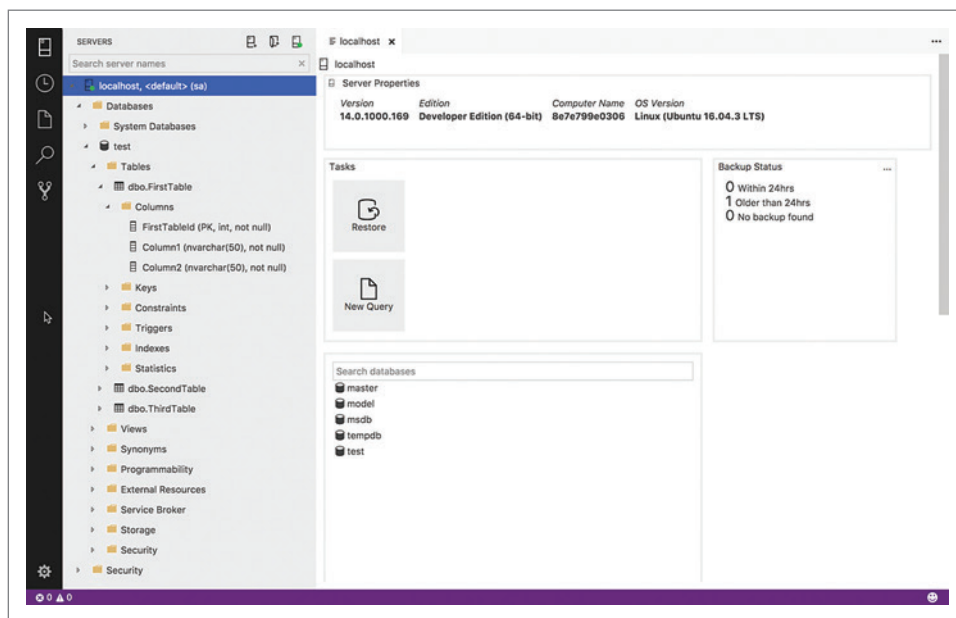


Figure 2 The Object Explorer and Servers Dashboard

only cross-platform, but also fill the needs of many types of users—DBAs, sys admins, accidental DBAs and developers.

VS Code provided a great starting point for SQL Operations Studio: It's cross-platform, highly extensible and written in ElectronJS, a platform for building desktop applications in JavaScript, HTML and CSS. In fact, if you're familiar with VS Code, you'll recognize the surface of SQL Operations Studio because of that VS Code starting point.

SQL Operations Studio debuted at the PASS Summit in October and a public preview was launched at Connect(). It's a free, standalone tool that works with Azure SQL Database, Azure SQL Data Warehouse and SQL Server running anywhere. You don't need a SQL Server license to use it. SQL Operations Studio is available for Linux, macOS and Windows at aka.ms/sqlopsstudio, and takes just moments to install. You'll find the source code for SQL Operations Studio on GitHub at github.com/microsoft/sqlopsstudio and anyone in the community can file issues or suggestions and contribute to the product on GitHub.

Just because it's cross-platform, I had to first try it out on my MacBook where I already have a few SQL Server for Linux servers running in Docker, but can also interact with some Azure SQL Databases or even connect to a SQL Server instance on a Windows Server on my network.

The SQL Operations Studio IDE

Let's take a look first at what you see when you start up SQL Operations Studio for the first time (see **Figure 1**). The arrows and text aren't part of the IDE, just there to help with the tour.

The action bar on the left, familiar to VS Code users, has 5 icons: Object Explorer, like its counterparts in SSMS and other database IDEs, lets you view and manage objects; Task History shows you the tasks, such as backup and restore, that have been performed;

File Explorer provides a way to store TSQL and other assets you want to associate with a particular project; Search here focuses on the files you're working with, but don't worry, there's also a way to search the database. Finally, Source Control leads to tools for the integrated source control of the project files.

Make Your First Server Connection

Like SSMS, SQL Operations Studio lets you connect to multiple servers and remembers the connections so you can easily reconnect whenever you open the application. I'll start by connecting to a SQL Server for Linux instance in a Docker container on my machine. First, I need to be sure the container is running, so

I'll open the integrated terminal window with CTRL+~ and type the Docker command, `docker ps`. The response tells me the `juliesqllinux` container is running and I know it's available at `localhost`.

Next, I'll click on the Object Explorer icon to open the Object Explorer, and then I'll click the Add Server icon at the top right.

This opens up a connection window that I'll populate with my connection information. As with SSMS, you can just add the basic connection information and let the defaults do the rest of the job. Or you can click Advanced and specify more information about the connection. I'll go with the easy option. After filling out the connection information, click Connect.

The server will appear in the Object Explorer and you'll be presented with some information about it in the dashboard. In **Figure 2**, I've partially expanded the server and one of the databases so you can see how much detail is available in Object Explorer. Knowing what was already possible, thanks to my experience with the mssql extension for VS Code (SQL Operations Studio includes all of those features), the addition of this explorer feature sold me instantly on SQL Operations Studio. But, of course, there's more.

Dashboards Provide Easy Access to Stats and Management Tasks

Servers and databases both have dashboards and you can customize what's on them (more on that shortly). If you right-click a database within the server database and choose Manage, you'll see that it also has a Backup Task button.

Returning to the icons at the top of Object Explorer, the middle one is for creating server groups. I'll click on that and use the simple form to create a group named DockerServers and assign it a color. Then I'll create a second one called AzureServers.

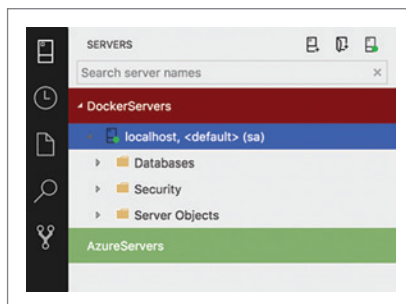


Figure 3 Server Groups in the Servers Explorer

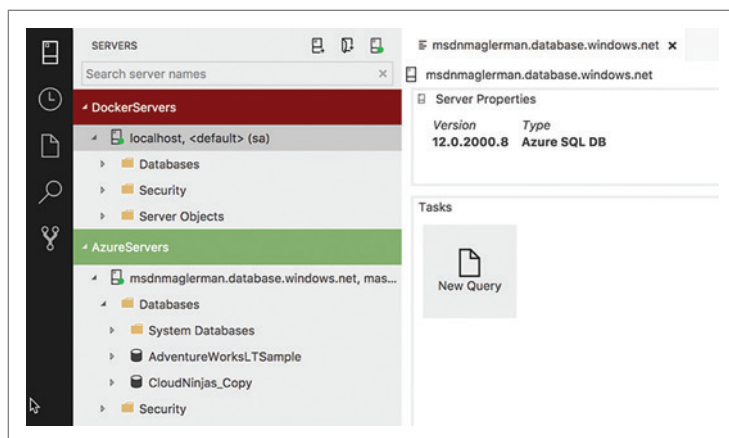


Figure 4 The Servers Dashboard for an Azure SQL Database Server—SQL Operations Studio Doesn't Show the Backup Task for Azure SQL Database Because Backups Are Automatic

Both of these groups will now show in the explorer window. I can drag the existing localhost onto the DockerServers Group bar and it will move it into that group, as shown in **Figure 3**.

Next, I'll create a connection to one of my Azure SQL servers. There are a few paths, but because I want it in the AzureServers group, I can click on the green bar and choose New Connection from its context menu. Back in the connection window, you'll see that the last option provided is Server group. That will be auto-populated with AzureServers and then I can fill out the rest of the information to connect to my Azure SQL Database server. After I've made the connection, I can see that server, its databases and the dashboard, which tells me it's an Azure SQL Database (see **Figure 4**). Notice that the Backup and Restore tasks aren't available. Azure SQL Database backs up data automatically, so there's no need to have an explicit task for triggering such operations. Restore works differently in Azure SQL Database, so I'm hoping to see a Restore task for Azure SQL Database in a future update.

Interacting with Data

Let's work with some data! I added the tried-and-true AdventureWorksLT database into my Azure SQL Database server because it's available as a sample when you create a new server. Another bonus is that it's prepopulated with lots of data. I'll do something I often do in SSMS—manually edit some data from a table. Expanding AdventureWorksLTSample again, I'll right-click on the SalesLT.Customer table in the explorer, which displays a context menu filled with functions, as shown in **Figure 5**.

I'll choose Edit Data to open up the grid shown in **Figure 6**, which has a Max Rows dropdown defaulted to 200 (a safe bet when pulling data over the Internet. The other options are 1,000 and 10,000.) I edited the Title in the first row, which was saved automatically when I moved to another cell. To see if the table really updated to Azure, I then opened a new query window (CMD+N) and was happy to have IntelliSense help me type SELECT* from SalesLT.Customer, prompting me

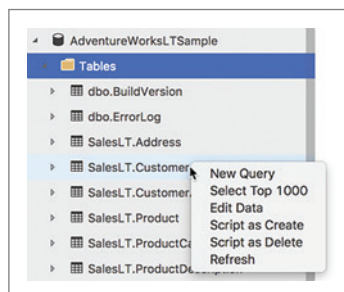


Figure 5 The Context Menu for a Database Table

for the available schemas and objects. I was also pleased to discover I could use the familiar F5 keystroke to execute queries—something you can't do in VS Code because so many tools and extensions have to share the keyboard shortcuts.

While you can edit data directly, you can't yet edit the database schema visually. The Script as Create menu item is the closest you can get in these early days of SQL Operations Studio to being able to modify schema in the database.

Customizing the Dashboard

Most of the features of the mssql extension for VS Code are in SQL Operations Studio. If you've used that extension already, or read my article (msdn.com/magazine/mt809115) or watched my Pluralsight course (bit.ly/PS_MSSQL), you may already be familiar with the many features of the SQL editor, as well as the available snippets for writing and executing queries and commands against a SQL Server database. The query results window, with its ability to export results to CSV, JSON or

Excel files, is another feature that came from the extension. In SQL Operations Studio, a new addition to the query results window allows you to tap into some of the amazing extensibility in this IDE.

Let me demonstrate and then do a big reveal of something I've never done in SSMS as I complete this little demo. In preparation for this functionality, it's time to work with the file system. Create a folder on your computer where you'll save some of the SQL you'll be writing.

Back in SQL Operations Studio, click the File Explorer icon in the Activity Bar and then the Open Folder button to open the folder you created. The File Explorer will display the folder and any files within (currently there are none).

You can create a new file inside the folder by clicking the "new file" icon to the right of the folder name in the File Explorer window. Be sure to give it a .sql extension. As shown in **Figure 7**, I've called mine TableSizes.sql.

I'm interested in some metadata about my AdventureWorksLT database: how many rows are in each of the tables and how much space are they taking up on my drive? Or, in this case, how much storage in my Azure account? Rather than spending hours trying to figure out how to write that query, thankfully there's a snippet that gives me just what I need, plus a bit more metadata along the same lines.

In the editor window, start by typing SQL and you'll see a list of the snippets. SQLGetSpaceUsed, shown in **Figure 7**, is the one I'm looking for. You can tab to auto-complete the snippet name and then hit enter to display the snippet's SQL in the editor window. The snippet has a placeholder to type in a table name, but I don't want to filter on a particular table; I want all of the tables. Scroll down to line 20 of the SQL and remove the following line:

```
WHERE TABL.name LIKE '%TableName%'
```

The query returns the following columns:

```
INDX.name AS index_name,
SUM(PART.rows) AS rows_count,
SUM(ALOC.total_pages) AS total_pages,
SUM(ALOC.used_pages) AS used_pages,
SUM(ALOC.data_pages) AS data_pages,
(SUM(ALOC.total_pages)*8/1024) AS total_space_MB,
(SUM(ALOC.used_pages)*8/1024) AS used_space_MB,
(SUM(ALOC.data_pages)*8/1024) AS data_space_MB
```


Spreadsheets Made Easy.



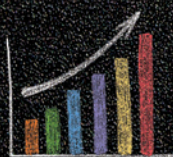
SpreadsheetGear 2017 Released

SpreadsheetGear 2017 adds a new SpreadsheetGear for .NET Standard product, official support for Excel 2013 and Excel 2016, 51 new Excel functions for a total of 449 fully supported functions, full conditional formatting support, enhanced workbook protection and encryption, cell gradient rendering and more.



Support for iOS, Android, Linux, macOS, UWP and more

SpreadsheetGear for .NET Standard enables cross-platform developers to enjoy the same high performance Excel-compatible reporting, charting, calculations and more relied on by thousands of Windows developers for 10+ years.



Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.

Download your free fully functional evaluation at SpreadsheetGear.com



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.



Windows
Forms



Silverlight



WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.



Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.



SpreadsheetGear

	CustomerID	NameStyle	Title	FirstName	MiddleName
1	1	0	Mr.	Orlando	N.
2	2	0	Mr.	Keith	NULL
3	3	0	Ms.	Donna	F.
4	4	0	Ms.	Janet	M.
5	5	0	Mr.	Lucy	NULL
6	6	0	Ms.	Rosmarie	J.
7	7	0	Mr.	Dominic	P.
8	10	0	Ms.	Kathleen	M.
9	11	0	Ms.	Katherine	NULL

Figure 6 Editing Data in SQL Operations Studio

I only want two pieces of data, the rowcount, now named Rows, and a twist on the data_space_MB. I've deleted all the other lines above and modified the two I want to keep:

```
SUM(PART.rows) AS Rows,
(SUM(ALLOC.total_pages)*8/1.024) AS Bytes
```

Notice that I changed the total space from megabytes to bytes by multiplying by 1.024 rather than 1,024. I have my reasons and you may agree when you see the results. Be sure to save the file. I've modified the VS Code files.AutoSave setting to save "AfterDelay" so I don't have to remember.

Now run the query and you'll be prompted to choose the connection on which to run it. If you don't specify a database, the query will be run against the master database. A handy recent connections list should make it easy to set the connection without having to fill the form out manually again. I could have created the new query window from the server, database or table and avoided having to select the connection for the query to run on, but then I would have had to specify where to save the .sql file. But because I happened to start with the file, I have to explicitly choose the connection. After selecting the connection, you'll see a grid of the query results displayed below the editor window.

To the right of the results, there are four icons. The first exports the results to a CSV file, the second to JSON, the third to Excel—all functionality that came over from the mssql extension. The fourth icon is new to SQL Operations Studio. It will create a graph from the data, with a variety of graph types to from which to choose. I was surprised by this feature—it's one that's reminiscent of sophisticated business intelligence (BI) tools. A horizontal graph suited my needs. In **Figure 8**, you can see the row count in pink and the bytes in blue. The reason I calculated bytes, not megabytes, is because the megabyte values would have been too small to appear on the chart.

That's already pretty cool, but wait, there's more! Copy as image and Save as image are great features, but they pale in comparison to the third option. Click the Create Insight button above the graph to open a new window with JSON that describes the graph in the form of a widget (see **Figure 9**). Widgets are a powerful feature of SQL Operations Studio that can help you create a lot of visual customizations.

Notice the queryfile value pointing to the .sql file I saved in my project folder. The widget knows to execute that query when it comes into view. If you make changes to the query in the file, those changes will be reflected in the widget the next time it's run. I want to tweak two things that I didn't modify in the Chart Viewer, which I can do directly in the JSON. I'll change the value of legendPosition to "top" and columnsAsLabels to "true."

The next task is to get this widget into the SQL Operation Studio settings. Copy the full text of the JSON file, then open the settings window again (CMD+.). On the left pane where the default settings are listed, use the search box to find "dashboard" and look within the results for dashboard.

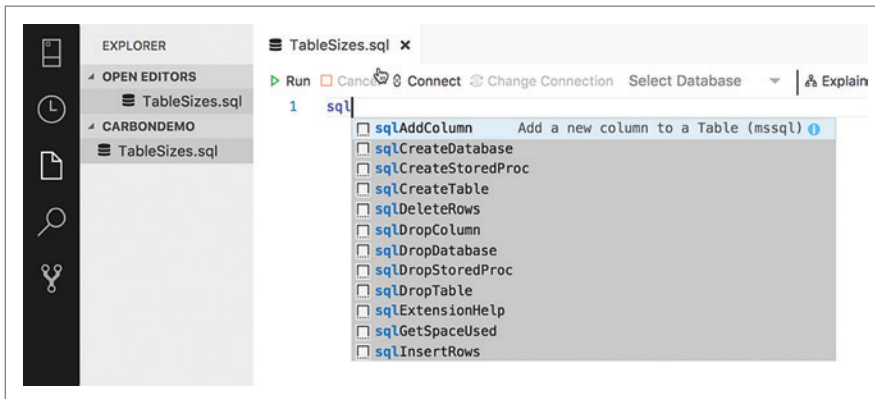


Figure 7 Snippets to Help with Tricky SQL Commands

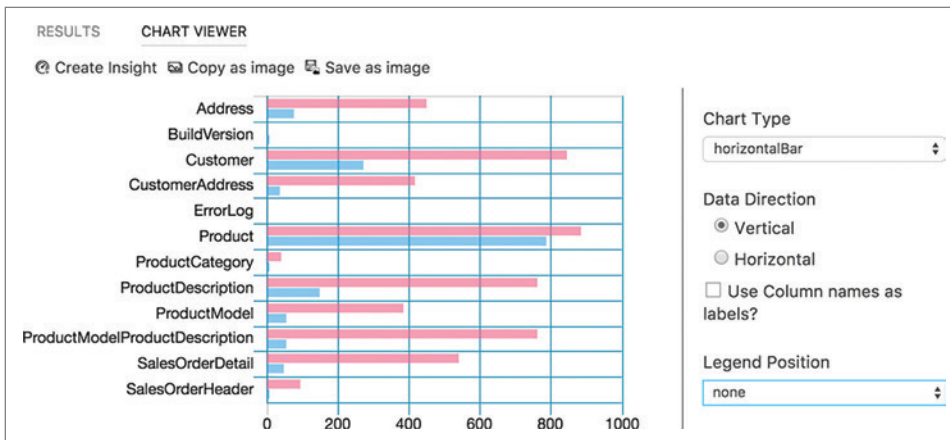


Figure 8 A Graph Generated from Query Results

Figure 9 The Widget Code Created from the Graph View of the Query Results

```
{
  "name": "My-Widget",
  "gridItemConfig": {
    "sizeX": 2,
    "sizeY": 1
  },
  "widget": {
    "insights-widget": {
      "type": {
        "horizontalBar": {
          "dataDirection": "vertical",
          "dataType": "number",
          "legendPosition": "none",
          "labelFirstColumn": false,
          "columnsAsLabels": false
        }
      }
    }
  },
  "queryFile": "/Users/julielerman/Documents/sqlpsstudio/TableSizes.sql"
}
```

database.widgets. Hover over that text and a pencil icon will appear to the left. Click the pencil icon and select the “Replace in settings” menu option that appears. This will cause the entire section of the default setting to be copied over to the user settings panel on the right. VS Code is so cool, isn’t it? Now, paste the JSON you copied above the opening brace for the Tasks widget, as shown here, and follow it with a comma:

```
"dashboard.database.widgets": [
  **paste your new widget here**
  {
    "name": "Tasks",
```

Save the settings file and close it. Then, back in Object Explorer, right-click on one of the databases and choose Manage to see its dashboard. The new widget appears on the dashboard (see **Figure 10**) and on the dashboard of every database you open. The query it’s tied to will run on-demand when you open the dashboard for a current view. This widget won’t appear on the Servers dashboard because I specifically placed it in the Database dashboard settings,

where it makes the most sense. But imagine the types of metadata you can expose visually on the dashboards to, for example, see which queries are running slowly or perform other health checks or view important statistics. You can also use the settings to control how widgets are laid out in the dashboard.

We’ve Just Scratched the Surface

There is so much more to discover and do in SQL Operations Studio. Here are a few extra tidbits before wrapping up.

You may have noticed the Explain button on the query window. It will show you query plans just as you see them in SSMS, with an alternate grid view, as well.

The file you created for the TableSize query can now be tracked and shared with the integrated source control. I’ve already been doing that in this project to save some queries that I spent too much time working out. If your team is using source control already for your databases, you’ll find many more sophisticated uses for this feature.

Take a look back at the dashboard in **Figure 1** and notice the search widget. A Server dashboard will show a list of its databases and you can easily search for database objects rather than perusing through the Object Explorer. The Database dashboard will show a list of tables, views, functions and procedures, and you can search for database objects by name there, as well.

You can learn so much more about SQL Operations Studio in the official docs at aka.ms/sqlpsstudio, where you’ll find detailed documents about its features, as well as walk-throughs. Remember that SQL Operations Studio was spawned from VS Code, which already has more than 4,500 extensions, most of which have come from the community. In addition to the enormous amount of work that the SQL Data Tools team is pouring into SQL Operations Studio, this new tool will likely take on a life of its own when its own ecosystem of extensions begins to evolve.

Head over to aka.ms/sqlpsstudio to download SQL Operations Studio for Linux, macOS and Windows and check out the Getting

Started guides. You can watch videos of SQL Operations Studio at aka.ms/sqlpsstudio-tutorial. And don’t forget to provide feedback, file issues, make suggestions and submit pull requests to improve SQL Operations Studio at github.com/microsoft/sqlpsstudio. ■

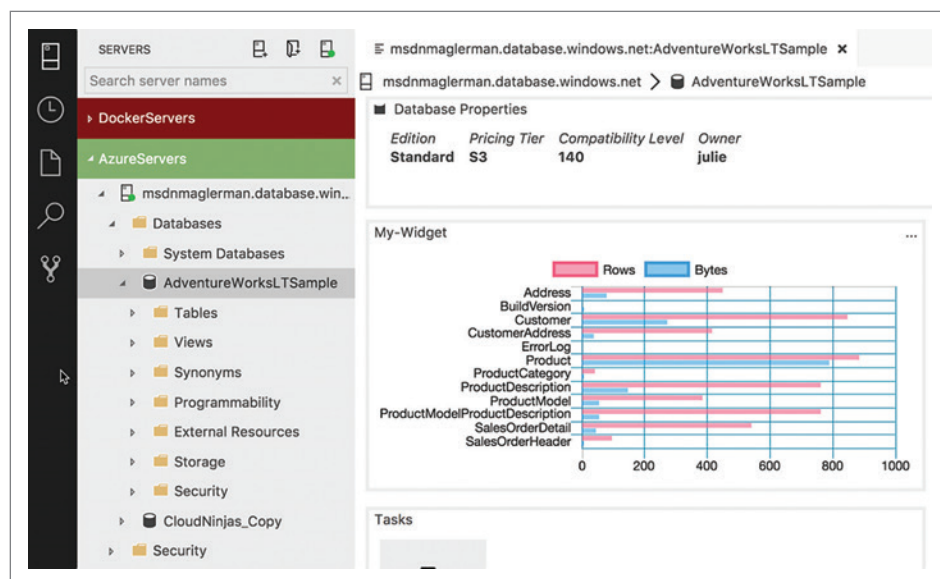


Figure 10 The Dashboard for the AdventureWorksLTSample Database with the New Widget in Place

JULIE LERMAN is a Microsoft Regional Director, Microsoft MVP, software team mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other topics at user groups and conferences around the world. She blogs at the.datafarm.com/blog and is the author of “Programming Entity Framework,” as well as a Code First and a DbContext edition, all from O’Reilly Media. Follow her on Twitter: [@julielerman](https://twitter.com/julielerman) and see her Pluralsight courses at julieme/PS-Videos.

THANKS to the following Microsoft technical experts for reviewing this article: Eric Kang and Sanjay Nagamangalam

Visual Studio[®] LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

March 11 – 16, 2018
Bally's Hotel & Casino
Las Vegas

Respect the Past. Code the Future.

Visual Studio Live! (VSLive!™) Las Vegas, returns to the strip, March 11 – 16, 2018. During this intense week of developer training, you can sharpen your skills in everything from ASP.NET to Xamarin.

Plus, celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Experience the education, knowledge-share and networking at #VSLive25.



VSLive! 1998



VSLive! 2017

SUPPORTED BY



PRODUCED BY



DEVELOPMENT TOPICS INCLUDE:



VS2017/.NET



Angular/JavaScript



ASP.NET Core



Xamarin



Azure / Cloud



Hands-On Labs



Software Practices



ALM / DevOps



SQL Server 2017



UWP (Windows)



Who Should Attend and Why

We've been around since 1993. What's our secret? YOU! Since our first conference (VBITS/ VSLive!/Visual Studio Live!), tens of thousands of developers, software architects, programmers, engineers, designers and more have trusted us year-in-and-year-out for unbiased and cutting-edge education on the Microsoft Platform.



Register to code with us today!

Register by January 19 and Save \$400!

Use Promo Code VSLJAN4

vslive.com/lasvegasmcdn

CONNECT WITH US



[@vslive](https://twitter.com/vslive)



facebook.com –
Search “VSLive”



linkedin.com – Join the
“Visual Studio Live” group!

BACK BY POPULAR DEMAND

Sunday Pre-Con Hands-On Labs

Choose From:

HOL01 Special 2-Day Hands-On Lab: Modern Security Architecture for ASP.NET Core

Sunday, March 11,
9:00am – 6:00pm (Part 1)*

Monday, March 12,
9:00am – 6:00pm (Part 2)*

Brock Allen

You will learn:

- The security architecture of ASP.NET Core
- About authenticating users with OpenID Connect
- How to protect Web APIs with OAuth2

*This 2-day Hands-On Lab is available with the six-day conference package or on its own. Details at vslive.com/lasvegasmndn.

HOL02 From 0-60 in a Day with Xamarin and Xamarin.Forms

Introductory / Intermediate

Sunday, March 11,
9:00am – 6:00pm

Roy Cornelissen & Marcel de Vries

You will learn:

- How to build your first mobile apps on three platforms with the Xamarin framework
- How to maintain platform uniqueness while sharing a large chunk of your codebase
- How to think “mobile first” in your application architecture

HOL03 Busy Developer's HOL on Angular

Sunday, March 11,
9:00am – 6:00pm

Ted Neward

In this Hands-On Lab, we'll start from zero, with a little TypeScript, then start working with Angular 2: its core constructs and how it works with components, modules, and of course the ubiquitous model/view/controller approach.

ONLY \$645 through January 19
Applies to HOL02 and HOL03 only.

ALM / DevOps		Cloud Computing	Database and Analytics	Native Client
START TIME	END TIME	Full Day Hands-On Labs: Sunday, March 11, 2018 <i>(Separate entry fee required)</i>		
8:00 AM	9:00 AM	Pre-Conference Hands-On Lab Registration - Coffee and Morning Pastries		
9:00 AM	6:00 PM	HOL01 Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 1) - Brock Allen		
4:00 PM	6:00 PM	Conference Registration Open		
START TIME	END TIME	Pre-Conference Workshops: Monday, March 12, 2018 <i>(Separate entry fee required)</i>		
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries		
9:00 AM	6:00 PM	HOL01 Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 2) - Brock Allen		
7:00 PM	9:00 PM	Dine-A-Round		
START TIME	END TIME	Day 1: Tuesday, March 13, 2018		
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries		
8:00 AM	9:15 AM	T01 Go Serverless with Azure Functions - Eric D. Boyd	T02 Getting Ready to Write Mobile Applications with Xamarin - Kevin Ford	
9:30 AM	10:45 AM	T05 Angular 101 - Deborah Kurata	T06 Lessons Learned from Real World Xamarin.Forms Projects - Nick Landry	
11:00 AM	12:00 PM	KEYNOTE: .NET Everywhere and for Everyone - James Montemagno, Principal		
12:00 PM	1:00 PM	Lunch		
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors		
1:30 PM	2:45 PM	T09 Busy Developer's Guide to Chrome Development - Ted Neward	T10 Works On My Machine... Docker for Developers - Chris Klug	
3:00 PM	4:15 PM	T13 Angular Component Communication - Deborah Kurata	T14 Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - Laurent Bugnion	
4:15 PM	5:30 PM	Welcome Reception		
START TIME	END TIME	Day 2: Wednesday, March 14, 2018		
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries		
8:00 AM	9:15 AM	W01 The Whirlwind Tour of Authentication and Authorization with ASP.NET Core - Chris Klug	W02 Building Mixed Reality Experiences for HoloLens & Immersive Headsets in Unity - Nick Landry	
9:30 AM	10:45 AM	W05 TypeScript: The Future of Front End Web Development - Ben Hoelting	W06 A Dozen Ways to Mess Up Your Transition From Windows Forms to XAML - Billy Hollis	
11:00 AM	12:00 PM	General Session: To Be Announced - Kasey Uhlenhuth, Program Manager, .NET &		
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch		
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)		
1:30 PM	1:50 PM	W09 Fast Focus: 0-60 for Small Projects in Visual Studio Team Services - Alex Mullans		
2:00 PM	2:20 PM	W12 Fast Focus: HTTP/2: What You Need to Know - Robert Boedigheimer		
2:30 PM	3:45 PM	W15 Advanced Fiddler Techniques - Robert Boedigheimer	W16 Building Cross-Platforms Business Apps with C# and CSLA .NET - Rockford Lhotka	
4:00 PM	5:15 PM	W19 Assembling the Web - A Tour of WebAssembly - Jason Bock	W20 Radically Advanced XAML: Dashboards, Timelines, Animation, and More - Billy Hollis	
7:00 PM	8:30 PM	VSLive! High Roller Evening Out		
START TIME	END TIME	Day 3: Thursday, March 15, 2018		
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries		
8:00 AM	9:15 AM	TH01 ASP.NET Core 2 For Mere Mortals - Philip Japikse	TH02 Performance in 60 Seconds – SQL Tricks Everybody MUST Know - Pinal Dave	
9:30 AM	10:45 AM	TH05 Getting to the Core of ASP.NET Core Security - Adam Tuliper	TH06 Secrets of SQL Server - Database Worst Practices - Pinal Dave	
11:00 AM	12:00 PM	Panel Discussion: To Be Announced		
12:00 PM	1:00 PM	Lunch		
1:00 PM	2:15 PM	TH09 Entity Framework Core 2 For Mere Mortals - Philip Japikse	TH10 SQL Server 2017 - Intelligence Built-in - Scott Klein	
2:30 PM	3:45 PM	TH13 MVVM and ASP.NET Core Razor Pages - Ben Hoelting	TH14 Introduction to Azure Machine Learning - James McCaffrey	
4:00 PM	5:15 PM	TH17 Securing Web Apps and APIs with IdentityServer - Brian Noyes	TH18 Introduction to the CNTK v2 Machine Learning Library - James McCaffrey	
START TIME	END TIME	Post-Conference Workshops: Friday, March 16, 2018 <i>(Separate entry fee required)</i>		
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries		
8:00 AM	5:00 PM	F01 Workshop: Creating Mixed Reality Experiences for HoloLens & Immersive Headsets with Unity - Nick Landry & Adam Tuliper		

Speakers and sessions subject to change

Software Practices	Visual Studio / .NET Framework	Web Client	Web Server
Full Day Hands-On Labs: Sunday, March 11, 2018 <small>(Separate entry fee required)</small>			
Pre-Conference Hands-On Lab Registration - Coffee and Morning Pastries			
HOL02 Full Day Hands-On Lab: From 0-60 in a Day with Xamarin and Xamarin.Forms - Roy Cornelissen & Marcel de Vries		HOL03 Full Day Hands-On Lab: Busy Developer's HOL on Angular - Ted Neward	
Conference Registration Open			
Pre-Conference Workshops: Monday, March 12, 2018 <small>(Separate entry fee required)</small>			
Pre-Conference Workshop Registration - Coffee and Morning Pastries			
M02 Workshop: Developer Dive into SQL Server 2016 - Leonard Lobel		M03 Workshop: Add Intelligence to Your Solutions with AI, Bots, and More - Brian Randell	
Dine-A-Round			
Day 1: Tuesday, March 13, 2018			
Registration - Coffee and Morning Pastries			
T03 Database Development with SQL Server Data Tools - Leonard Lobel		T04 What's New in Visual Studio 2017 for C# Developers - Kasey Uhlenhuth	
T07 Introduction to Azure Cosmos DB - Leonard Lobel		T08 Using Visual Studio Mobile Center to Accelerate Mobile Development - Kevin Ford	
Program Manager – Xamarin, Microsoft			
Lunch			
Dessert Break - Visit Exhibitors			
T11 DevOps for the SQL Server Database - Brian Randell		T12 To Be Announced	
T15 PowerShell for Developers - Brian Randell		T16 To Be Announced	
Welcome Reception			
Day 2: Wednesday, March 14, 2018			
Registration - Coffee and Morning Pastries			
W03 Using Feature Toggles to Separate Releases from Deployments - Marcel de Vries		W04 Lock the Doors, Secure the Valuables, and Set the Alarm - Eric D. Boyd	
W07 Overcoming the Challenges of Mobile Development in the Enterprise - Roy Cornelissen		W08 Computer, Make It So! - Veronika Kolesnikova & Willy Ci	
Visual Studio, Microsoft			
Birds-of-a-Feather Lunch			
Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)			
W10 Fast Focus: Cross Platform Device Testing with xUnit - Oren Novotny		W11 Fast Focus: Understanding .NET Standard - Jason Bock	
W13 Fast Focus: Serverless Computing: Azure Functions and Xamarin in 20 minutes - Laurent Bugnion		W14 Fast Focus: TBD - Scott Klein	
W17 Versioning NuGet and npm Packages - Alex Mullans		W18 Getting to the Core of .NET Core - Adam Tuliper	
W21 Encrypting the Web - Robert Boedigheimer		W22 Porting MVVM Light to .NET Standard: Lessons Learned - Laurent Bugnion	
VSLive! High Roller Evening Out			
Day 3: Thursday, March 15, 2018			
Registration - Coffee and Morning Pastries			
TH03 Demystifying Microservice Architecture - Miguel Castro		TH04 Cognitive Services in Xamarin Applications - Veronika Kolesnikova	
TH07 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - Jeremy Clark		TH08 Publish Your Angular App to Azure App Services - Brian Noyes	
Panel Discussion: To Be Announced			
Lunch			
TH11 Writing Testable Code and Resolving Dependencies - DI Kills Two Birds with One Stone - Miguel Castro		TH12 Signing Your Code the Easy Way - Oren Novotny	
TH15 "Doing DevOps" as a Politically Powerless Developer - Damian Brady		TH16 Analyzing Code in .NET - Jason Bock	
TH19 I'll Get Back to You: Task, Await, and Asynchronous Methods - Jeremy Clark		TH20 Multi-targeting the World: A Single Project to Rule Them All - Oren Novotny	
Post-Conference Workshops: Friday, March 16, 2018 <small>(Separate entry fee required)</small>			
Post-Conference Workshop Registration - Coffee and Morning Pastries			
F02 Workshop: Distributed Cross-Platform Application Architecture - Jason Bock & Rockford Lhotka		F03 Workshop: UX Design for Developers: Basics of Principles and Process - Billy Hollis	

Bally's Hotel & Casino will play host to Visual Studio Live!, and is offering a special reduced room rate to conference attendees.



CONNECT WITH
VISUAL STUDIO LIVE!



twitter.com/vslive –
@VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!



vslive.com/lasvegasmsdn

Introducing the Windows Compatibility Pack for .NET Core

Immo Landwerth

The **Microsoft .NET Framework** is still the best choice for certain styles of apps, especially for desktop apps and Web apps that use ASP.NET Web Forms. But if you need highly scalable Web apps, create self-contained deployments using Docker, or if you need to run on Linux, then you want to consider porting to .NET Core. But bringing existing code to .NET Core can be a challenge. In this article, I'll explain how you can use the new Windows Compatibility Pack for .NET Core. It provides access to APIs that were previously available only for .NET Framework (for example, System.Drawing, System.DirectoryServices, ODBC, WMI and many more). Because this includes both cross-platform and Windows-only technologies, it's critical to understand early if you're using APIs that might interfere with your cross-platform goals. I'll address this by showcasing the new API analyzer, which gives you live feedback as you're editing code.

Overview

When we shipped .NET Core 1.x, as well as .NET Standard 1.x, we were hoping to be able to use this as an opportunity to remove legacy technologies and deprecated APIs. Since then, we've learned

that no matter how attractive the new APIs and capabilities of .NET Core are, if the existing code base is large enough, the benefits of the new APIs are often dwarfed by the sheer cost of reimplementing or adapting that code. Luckily, the Windows Compatibility Pack provides a good chunk of these technologies so that building .NET Core applications and .NET Standard libraries becomes much more viable for existing code.

This capability is a continuation of .NET Standard 2.0 (bit.ly/2iuekmN), in which we significantly increased the number of APIs that can be shared across all .NET implementations, especially .NET Core. The goal was to make porting existing code much easier and ensure it largely compiles just as is. However, we also didn't want to complicate .NET Standard by adding large APIs that can't work across all platforms, which is why we haven't added the Windows registry or reflection-emit APIs. Because the Windows Compatibility Pack is a separate NuGet package and sits above .NET Standard, it's free to provide access to technologies that are Windows-only.

Providing more APIs for class libraries that target .NET Standard also helps with the compatibility mode we've added in .NET Standard 2.0. This compatibility mode allows the referencing of existing .NET Framework binaries, which helps with the transition period where many packages aren't yet available for .NET Standard or .NET Core. But this compatibility mode doesn't change physics: It can only bridge differences in assembly names between .NET Framework and .NET Standard. It can't give you access to APIs that don't exist for the .NET implementation on which you're running. For example, if you're referencing a .NET Framework library that uses System.DirectoryServices, it will fail if you run it on .NET Core 2.0 today, because System.DirectoryServices isn't included in .NET Core. The Windows Compatibility Pack helps to extend the set of APIs covered by the compatibility mode by bringing in System.DirectoryServices.

This article discusses:

- Migrating to .NET Core
- Using helpful tools: API Port, Windows Compatibility Pack, API Analyzer
- Understanding Windows-only dependencies

Technologies discussed:

.NET Core, .NET Standard, API Port, Windows Compatibility Pack, API Analyzer

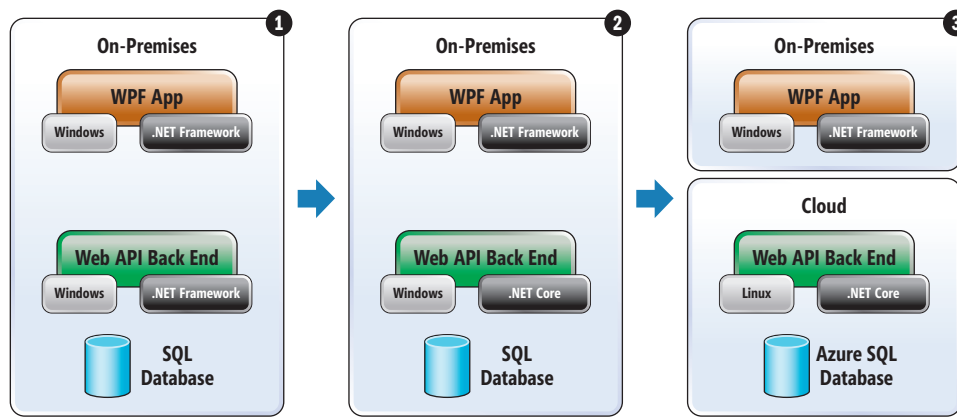


Figure 1 Migrating a Typical .NET App Partially to the Cloud

Plan Your Migration

Unless your project is very small, you should take the time to plan your migration, and the most important part is understanding what you want to get out of it. Moving to .NET Core just because it's the new hotness isn't a good enough reason (unless you're a true fan) as migrations are never free.

Let's look at a typical .NET app, the Fabrikam Asset Management application. It consists of a Windows Presentation Foundation (WPF) front end and a Web API back end, storing data in a SQL Server, all deployed on Windows and on-premises. Fabrikam decided it would like to move its back end to Azure. The company is quite happy with the desktop application and wants to continue to leverage WPF. It also decided it's best to use ASP.NET Core for the Web API back end as this allows more flexibility in the choice of server OS, as well as for isolated deployments using Docker.

A migration plan should include several steps instead of doing everything in one big swoop (also known as the big outage). This ensures you can deliver incremental value, keep your system operational, and learn and adapt as you perform the migration. Fabrikam's migration plan looks like what's shown in **Figure 1**.

As a first step, the plan moves the Web API back end to .NET Core, but the app will remain on Windows. This minimizes the amount of change the code must accommodate. The next step entails moving the .NET Core back end to the cloud. Then the company plans to move the back end to Linux. Later steps might involve Fabrikam deciding to leverage Docker.

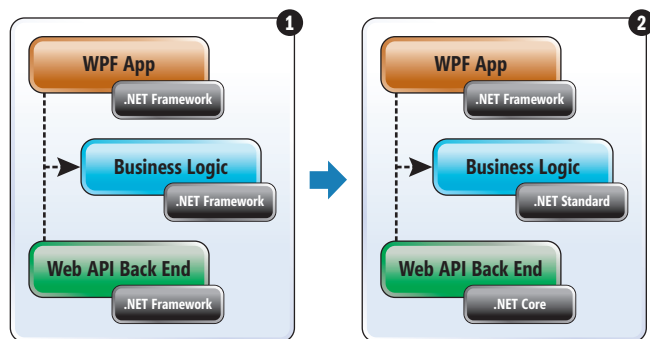


Figure 2 Handling Shared Code When Targeting Multiple .NET Implementations

The important point is: This is a step-by-step process and you want to make sure your application is operational after performing each step. This includes having the code compile and passing all tests (you do have tests, right?), but it might also mean being able to deploy the current system to production. Whether you need that is a function of how much time the migration will take and how much your system is under active development. Alternatively, you might decide to bring up the new system independently of the

old one to reduce the risk of operational disruption in case you manage to corner yourself.

Understanding your goals and migration path helps to reveal any new constraints you need to factor in when refining your architecture. In this case, Fabrikam needs to share business logic and infrastructure code between the WPF application and the Web API back end. In the current system, both are running on top of .NET Framework so that code is simply contained in a class library that also targets .NET Framework. Moving forward, Fabrikam needs to share that code between .NET Framework and .NET Core, so it decides to give .NET Standard 2.0 a shot, as shown in **Figure 2**.

Once you understand which part of your existing code needs to be ported and what it needs to be ported to, you should use the API Port tool (which you'll find at aka.ms/apiport). See my demo at bit.ly/2zkaKDn to understand how easy or difficult this might be. API Port scans your existing application binaries, including any third-party code you might have, and produces a report that shows you assembly by assembly how portable each is, and provides a table of all the APIs that are either unavailable or must be migrated. API Port isn't specific to .NET Standard or .NET Core: You can select any .NET implementation you want, including .NET Framework, .NET Core, .NET Standard, UWP, Mono and Xamarin. This allows you to plan your migrations regardless of what you're porting from and what you need to port to.

When you run API Port, I recommend you use the targets .NET Standard + Platform Extensions and .NET Core + Platform Extensions. Including the extensions ensures you don't get false negatives for APIs that don't ship as part of the platform, but can be added by referencing an additional NuGet package. I also recommend you use the command-line version of API Port and run it over your existing application as this is much easier, especially when your source code is spread across several solutions. This also allows you to assess third-party dependencies.

In the Fabrikam case, however, the company only needs to run API Port over the shared library:

```
$ apiport analyze -f C:\src\fabrikam\bin\Fabrikam.Shared.dll -t ".NET Standard + Platform Extensions"
```

Fortunately for Fabrikam, the report API Port produces shows its library only uses APIs that are available for .NET Standard 2.0 (see **Figure 3**).

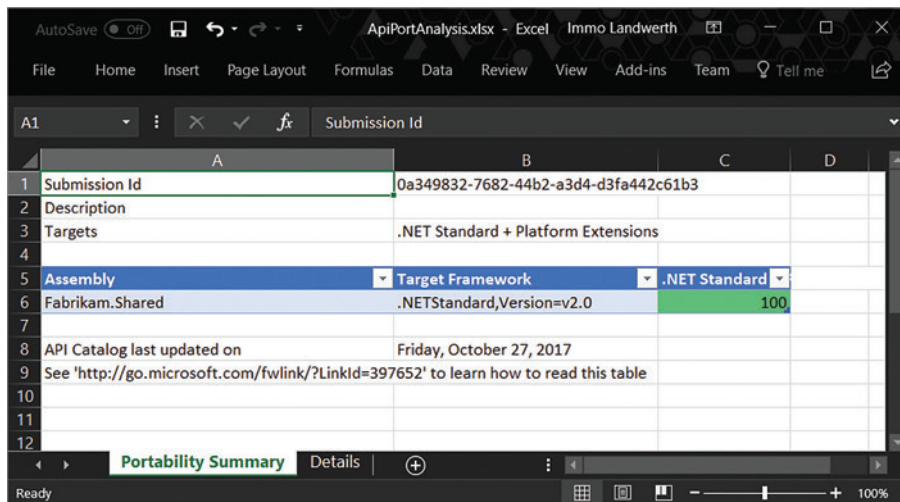


Figure 3 API Port Results for Fabrikam.Shared

What's in the Compatibility Pack?

The Windows Compatibility Pack is represented by the Microsoft.Windows.Compatibility NuGet package and contains about 40 components (see **Figure 4** for the full list).

Given that .NET Framework was designed for Windows, about half of the compatibility pack is Windows-only as it depends on or wraps Windows technologies. But, as discussed earlier, the first step in migrating an existing .NET Framework code base should be to move to .NET Core but remain on Windows. For that step, not being able to use Windows-only technologies would just be a migration hurdle with zero architectural benefit.

the full list of components without having to install dozens of packages. However, as you're migrating and removing dependencies on legacy technologies, it can be useful to reference only the components you need and remove the ones you've migrated away from to make sure new code doesn't take a dependency on them again.

Using the Compatibility Pack

Let's look further at Fabrikam's Asset Management system. The infrastructure and business logic component needs to be shared between .NET Framework and .NET Core, which is why the company plans to convert it to target .NET Standard. After converting

Figure 4 Available and Upcoming Components in the Windows Compatibility Pack

Component	Status	Windows-Only	Component	Status	Windows-Only
Microsoft.Win32.Registry	Available	Yes	System.Management	Coming	Yes
Microsoft.Win32.Registry.AccessControl	Available	Yes	System.Runtime.Caching	Coming	
System.CodeDom	Available		System.Security.AccessControl	Available	Yes
System.ComponentModel.Composition	Coming		System.Security.Cryptography.Cng	Available	Yes
System.Configuration.ConfigurationManager	Available		System.Security.Cryptography.Pkcs	Available	Yes
System.Data.DatasetExtensions	Coming		System.Security.Cryptography.ProtectedData	Available	Yes
System.Data.Odbc	Coming		System.Security.Cryptography.Xml	Available	Yes
System.Data.SqlClient	Available		System.Security.Permissions	Available	
System.Diagnostics.EventLog	Coming	Yes	System.Security.Principal.Windows	Available	Yes
System.Diagnostics.PerformanceCounter	Coming	Yes	System.ServiceModel.Duplex	Available	
System.DirectoryServices	Coming	Yes	System.ServiceModel.Http	Available	
System.DirectoryServices.AccountManagement	Coming	Yes	System.ServiceModel.NetTcp	Available	
System.DirectoryServices.Protocols	Coming		System.ServiceModel.Primitives	Available	
System.Drawing	Coming		System.ServiceModel.Security	Available	
System.Drawing.Common	Available		System.ServiceModel.Syndication	Coming	
System.IO.FileSystem.AccessControl	Available	Yes	System.ServiceProcess.ServiceBase	Coming	Yes
System.IO.Packaging	Available		System.ServiceProcess.ServiceController	Available	Yes
System.IO.Pipes.AccessControl	Available	Yes	System.Text.Encoding.CodePages	Available	Yes
System.IO.Ports	Available	Yes	System.Threading.AccessControl	Available	Yes

Figure 5 Looking for the Registry, the Web API Crashes on Startup

```

Unhandled Exception: System.TypeInitializationException: The type
initializer for 'Microsoft.Win32.Registry' threw an exception. --> System.
PlatformNotSupportedException: Registry is not supported on this platform.
  at Microsoft.Win32.RegistryKey.OpenBaseKeyCore(RegistryHive hKey,
    RegistryView view)
  at Microsoft.Win32.Registry..cctor()
  --- End of inner exception stack trace ---
  at Fabrikam.Infrastructure.Logging.Logger.GetLoggingPath()
  at Microsoft.AspNetCore.Hosting.Internal.WebHost.EnsureStartup()
  at Microsoft.AspNetCore.Hosting.Internal.WebHost.EnsureApplicationServices()
  at Microsoft.AspNetCore.Hosting.Internal.WebHost.BuildApplication()
  at Microsoft.AspNetCore.Hosting.WebHostBuilder.Build()
  at Fabrikam.AssetManagement.WebApi.Program.BuildWebHost(String[] args)
  at Fabrikam.AssetManagement.WebApi.Program.Main(String[] args)

```

the project, the code no longer compiles because it uses APIs that aren't part of .NET Standard.

In this case, the logging component reads settings from the registry to determine where the logs should be placed:

```

private static string GetLoggingPath()
{
    using (var key = Registry.CurrentUser.OpenSubKey(
        @"Software\Fabrikam\AssetManagement"))
    {
        if (key?.GetValue("LoggingDirectoryPath") is string configuredPath)
            return configuredPath;
    }

    var appDataPath = Environment.GetFolderPath(
        Environment.SpecialFolder.LocalApplicationData);
    return Path.Combine(appDataPath, "Fabrikam", "AssetManagement", "Logging");
}

```

That's because the WPF desktop application happens to store its settings in the registry. Ideally, Fabrikam would refactor the application and would no longer store settings in the registry, but it would like to do this as a separate work item and not block the migration. The easiest way to get unblocked is by adding the compatibility pack. To do this, the company installs Microsoft.Windows.Compatibility. After the package is installed, the code compiles again. Fabrikam validates that both the WPF desktop application and the Web API continue to work with the newly created .NET Standard library.

At this point, you might wonder what happens if this code runs on Linux. So, let's do a fast-forward and see what happens when Fabrikam completes the migration of the Web API back end to .NET Core and starts to migrate from Windows to Linux. During startup, the Web API app configures, among other things, the logging infrastructure. As you might expect, that doesn't work so well because it calls into the registry to determine the path where the logs should be placed. And, sure enough, the Web API crashes on startup with an exception, as shown in Figure 5.

The fix for this is simple: Fabrikam needs to guard the registry call with a platform check, as shown in Figure 6.

In this case, the fix was straight-forward as the code already handled the situation where the registry didn't contain a configuration for the logging path. In general, you need to decide what the fallback logic is when you

Figure 6 Guarding the Registry Call

```

private static string GetLoggingPath()
{
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        using (var key = Registry.CurrentUser.OpenSubKey(
            @"Software\Fabrikam\AssetManagement"))
        {
            if (key?.GetValue("LoggingDirectoryPath") is string configuredPath)
                return configuredPath;
        }
    }

    var appDataPath = Environment.GetFolderPath(
        Environment.SpecialFolder.LocalApplicationData);
    return Path.Combine(appDataPath, "Fabrikam", "AssetManagement", "Logging");
}

```

can't call a Windows-only API. This might mean using a Linux-specific API, replacing the Windows-only concept with one that's cross-platform, or simply doing nothing. The latter can make sense if you just want to provide some extra value for customers who are running on Windows, as in this case.

Understanding Windows-Only Dependencies

Something tells me that reading the last paragraphs made you slightly uneasy. I bet you're wondering how you can use the Compatibility Pack safely in code you plan to migrate later or in code you know will need to run on Linux. You might even go so far as to conclude that this whole idea is diluting the promise of .NET Standard, which is about building libraries that can work on any .NET implementation and on any OS.

We don't think it's that dire because:

Being able to call non-portable APIs is convenient and powerful. At first, it's tempting to think you're better off if these APIs wouldn't be available at all. But consider the example of Fabrikam's shared library: It's much easier to just call the API under a guard than it is having to split, especially if your library needs just a very few Windows-only APIs. Of course, it might still make sense to refactor your code to not depend on platform-specific APIs—it just means you have the choice not to, if only temporarily.

Non-portable APIs aren't available by default. Having many Windows-only APIs is the primary reason why the APIs in Compatibility Pack aren't part of .NET Standard. We want to make sure that what .NET Standard offers by default works everywhere.

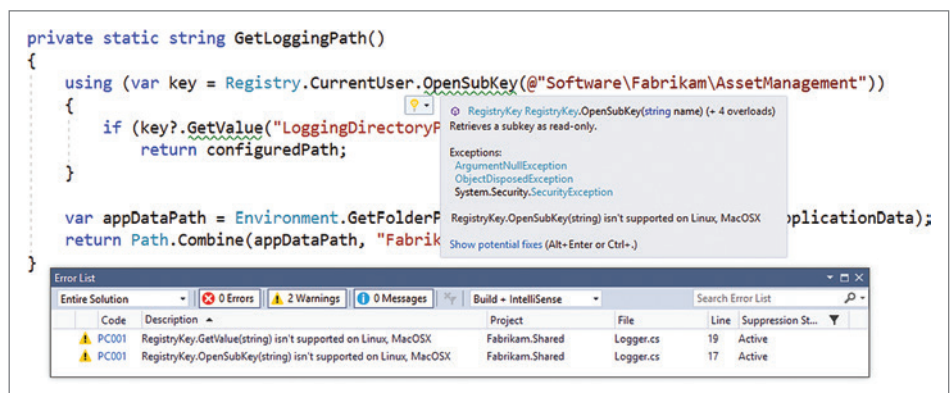


Figure 7 API Analyzer Warns When Non-Portable APIs Are Used



Figure 8 Suppressing Warnings for Use of Non-Portable APIs

Tooling warns when non-portable APIs are used. We provide an API Analyzer (bit.ly/2W9n7o) that detects usage of non-portable APIs and warns you as you're developing your code.

Let's see how API Analyzer would work in the context of Fabrikam's shared library. The analyzer is provided in the NuGet package `Microsoft.DotNet.Analyzers.Compatibility`. After it's installed, the error list shows warnings in the `GetLoggingPath` method, as shown in **Figure 7**.

It informs you that both `OpenSubKey` and `GetValue` aren't available and will throw `PlatformNotSupportedException` on Linux and macOS. It's worth pointing out that this analyzer is powered by Roslyn, which means it provides immediate feedback and doesn't require you to compile. This is very powerful as you're getting information as you're developing your code, so you can focus on the task at hand without having to remember to run a tool after the fact.

To address the warning, you'll need to add the platform guard and then suppress the warning, as shown in **Figure 8**.

My preference is to use the global suppression file (which will suppress this warning for the method you're currently in but records that suppression in a secondary file) as opposed to using the in-source option (which will put a `#pragma` suppression around the current statement), but the choice is yours. While the analyzer provides a great interactive experience in Visual Studio,

it's by no means tied to Visual Studio or to the concept of an IDE. Analyzers are run by the compiler. If you checked in the code and submitted it to your Linux CI server or just compiled it from the command line, you'd get the same warnings:

```
$ dotnet build
Logger.cs(17,43): warning PC001: RegistryKey.OpenSubKey(string) isn't
supported on Linux, MacOSX
Logger.cs(19,18): warning PC001: RegistryKey.GetValue(string) isn't
supported on Linux, MacOSX
Fabrikam.Shared -> /home/immol/Fabrikam.Shared/bin/Debug/netstandard2.0/
Fabrikam.Shared.dll

Build succeeded.
```

The analyzer is also configurable as to whether these are mere suggestions, warnings or even errors. The default is warnings, but, as **Figure 9** shows, you can configure them to be errors, too.

Wrapping Up

Each migration is different. Before porting to .NET Core, you should understand your goals and what you want the migration to accomplish. Use the API Port tool to assess how portable your existing code is.

Plan your migration not as a single monolithic operation but rather as a series of steps along the migration path. Don't assume you can port an existing application all at once. This allows you to realize value as you go and learn more about potential issues and how your architecture needs to change to become cross-platform.

Take advantage of the Windows Compatibility Pack, which provides access to about 40 .NET Framework components and can be referenced from both .NET Core and .NET Standard projects. However, definitely keep in mind that this package includes Windows-only components. If you plan to go cross-platform, use the API Analyzer, which will inform you whenever you're using non-portable APIs. Guard the calls to these APIs accordingly and provide sensible fallback logic, then suppress the warning.

And, last, keep in mind that not porting is also an option. The .NET Framework is still the best choice for certain kinds of apps, particularly desktop applications.

IMMO LANDWERTH is a program manager working on the .NET platform team at Microsoft. He specializes in the base class library, API design and .NET Standard.

THANKS to the following Microsoft technical experts who reviewed this article: Wes Haggard and Dan Moseley

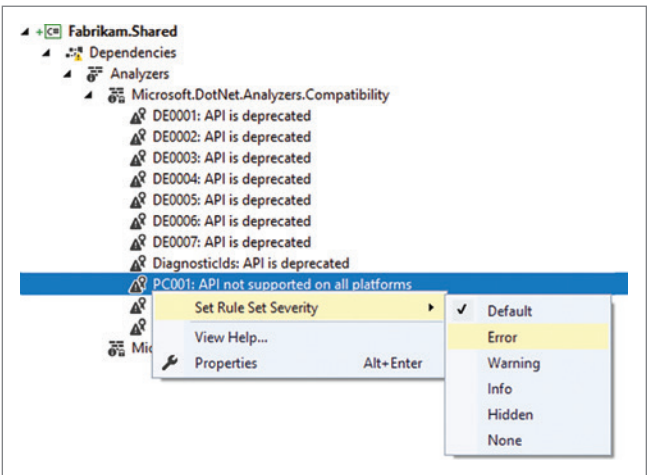
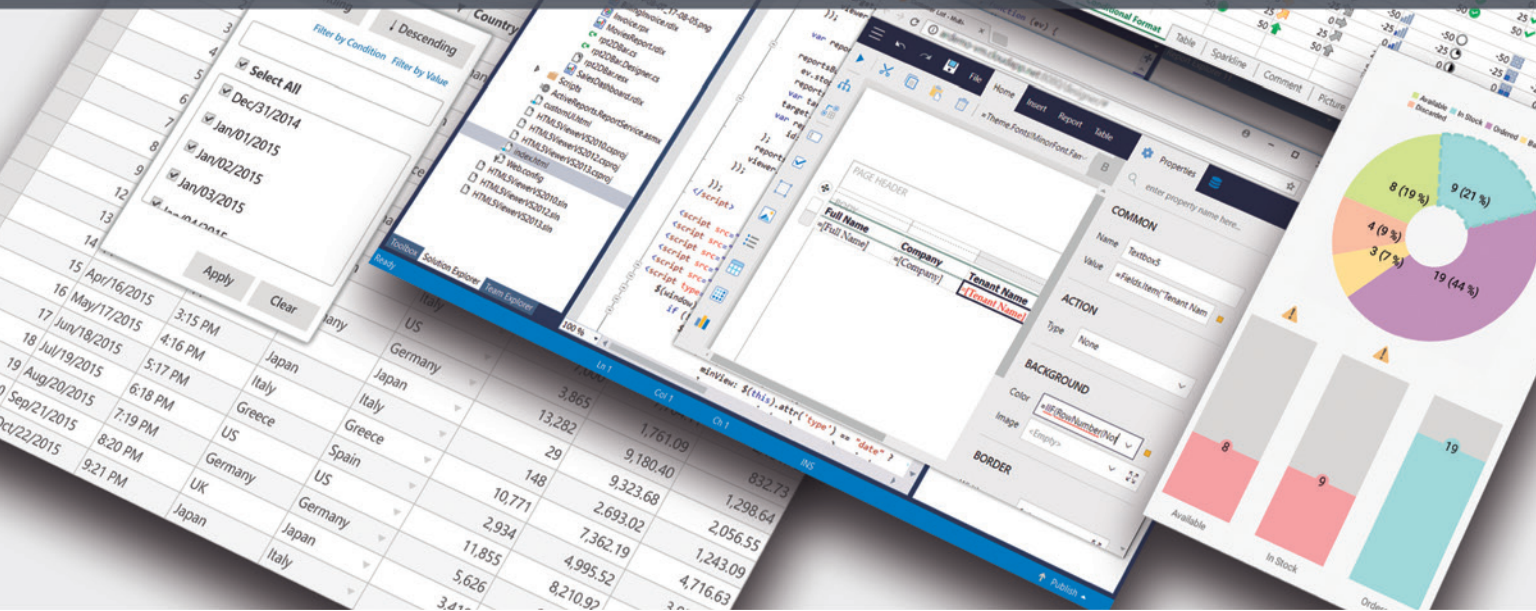


Figure 9 Configuring the Severity Level for Non-Portable APIs

Empower your development. Build better apps.

GrapeCity's family of products provides **developers, designers, and architects** with the ultimate collection of easy-to-use tools for building **sleek, high-performing, feature-complete** applications. With over 25 years of experience, we understand your needs and offer the industry's best support. **Our team is your team.**



ComponentOne
.NET UI CONTROLS



ActiveReports
REPORTING SOLUTIONS



Spread
SPREADSHEET SOLUTIONS



Wijmo
JAVASCRIPT UI CONTROLS

Learn more and get free 30-day trials at **GrapeCity.com**

For more information: **1.800.858.2739**



SYNCFUSION *SUCCINCTLY* SERIES

- 130+ e-books on popular technologies for free
- Ad-free
- 100 pages or less
- PDF and Kindle formats

